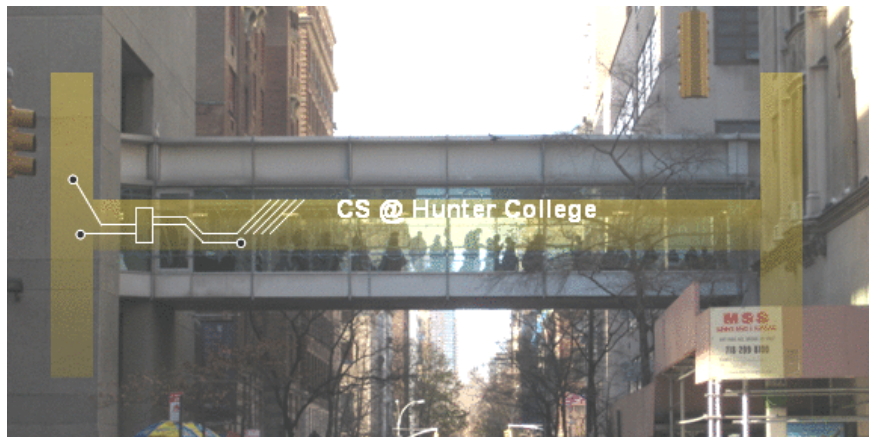


CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

- This lecture will be recorded

Announcements

- Thanksgiving Break starts in 9 days.



Announcements



- Thanksgiving Break starts in 9 days.
- No CUNY classes:
Thursday-Saturday, 26-29 November.

Announcements



- Thanksgiving Break starts in 9 days.
- No CUNY classes:
Thursday-Saturday, 26-29 November.
- Add my email to your contacts and check your spam folders. I reply within 24/48 hours at most (not on weekends).

Announcements



- Thanksgiving Break starts in 9 days.
- No CUNY classes:
Thursday-Saturday, 26-29 November.
- Add my email to your contacts and check your spam folders. I reply within 24/48 hours at most (not on weekends).
- In response to wrap-up requests, additional challenges today with while loops and binary & hexadecimal numbers.

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**
No worries– we'll start off with those in today's lecture.

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**
No worries– we'll start off with those in today's lecture.
- **When is the final? Is there a review sheet?**

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**
No worries– we'll start off with those in today's lecture.
- **When is the final? Is there a review sheet?**
The official final is Monday, 14 December, 9-11am.

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**
No worries– we'll start off with those in today's lecture.
- **When is the final? Is there a review sheet?**
The official final is Monday, 14 December, 9-11am.
The early final exam (alternative date) is on Friday, 11 December (Reading Day), 8-10am.

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**

No worries— we'll start off with those in today's lecture.

- **When is the final? Is there a review sheet?**

The official final is Monday, 14 December, 9-11am.

The early final exam (alternative date) is on Friday, 11 December (Reading Day), 8-10am.

Instead of a review sheet, we have:

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**

No worries– we'll start off with those in today's lecture.

- **When is the final? Is there a review sheet?**

The official final is Monday, 14 December, 9-11am.

The early final exam (alternative date) is on Friday, 11 December (Reading Day), 8-10am.

Instead of a review sheet, we have:

- ▶ *All previous final exams (and answer keys) on the website.*

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**

No worries— we'll start off with those in today's lecture.

- **When is the final? Is there a review sheet?**

The official final is Monday, 14 December, 9-11am.

The early final exam (alternative date) is on Friday, 11 December (Reading Day), 8-10am.

Instead of a review sheet, we have:

- ▶ *All previous final exams (and answer keys) on the website.*
- ▶ *UTAs in drop-in tutoring happy to review concepts and old exam questions.*

Frequently Asked Questions

From email and tutoring.

- **Help! Binary & hexadecimal numbers make no sense!**

No worries– we'll start off with those in today's lecture.

- **When is the final? Is there a review sheet?**

The official final is Monday, 14 December, 9-11am.

The early final exam (alternative date) is on Friday, 11 December (Reading Day), 8-10am.

Instead of a review sheet, we have:

- ▶ *All previous final exams (and answer keys) on the website.*
- ▶ *UTAs in drop-in tutoring happy to review concepts and old exam questions.*
- ▶ *There will be opportunity for some practice and to ask review questions during our last meeting on 8 December.*

Today's Topics



- Design Patterns: Searching
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

Today's Topics



- **Design Patterns: Searching**
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

Predict what the code will do:

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

Python Tutor

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

(Demo with pythonTutor)

Design Pattern: Linear Search

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

- Example of **linear search**.

Design Pattern: Linear Search

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

- Example of **linear search**.
- Start at the beginning of the list.

Design Pattern: Linear Search

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

- Example of **linear search**.
- Start at the beginning of the list.
- Look at each item, one-by-one.

Design Pattern: Linear Search

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

- Example of **linear search**.
- Start at the beginning of the list.
- Look at each item, one-by-one.
- Stopping, when found, or the end of list is reached.

Today's Topics



- Design Patterns: Searching
- **Python Recap**
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

Python & Circuits Review: 10 Weeks in 10 Minutes



A whirlwind tour of the semester, so far...

Week 1: print(), loops, comments, & turtles

Week 1: print(), loops, comments, & turtles

- Introduced comments & print():

```
#Name: Thomas Hunter
```

← *These lines are comments*

```
#Date: September 1, 2017
```

← *(for us, not computer to read)*

```
#This program prints: Hello, World!
```

← *(this one also)*

```
print("Hello, World!")
```

← *Prints the string "Hello, World!" to the screen*

Week 1: print(), loops, comments, & turtles

- Introduced comments & print():

```
#Name: Thomas Hunter
```

← These lines are comments

```
#Date: September 1, 2017
```

← (for us, not computer to read)

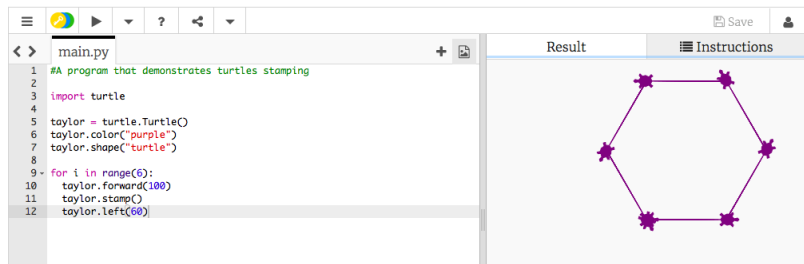
```
#This program prints: Hello, World!
```

← (this one also)

```
print("Hello, World!")
```

← Prints the string "Hello, World!" to the screen

- As well as definite loops & the turtle package:



The screenshot shows a Python IDE with a code editor on the left and a result window on the right. The code in the editor is as follows:

```
1 #A program that demonstrates turtles stamping
2
3 import turtle
4
5 taylor = turtle.Turtle()
6 taylor.color("purple")
7 taylor.shape("turtle")
8
9 for i in range(6):
10     taylor.forward(100)
11     taylor.stamp()
12     taylor.left(60)
```

The result window displays a purple hexagon with a turtle shape at each of its six vertices.

Week 2: variables, data types, more on loops & range()

Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.

Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers

Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers
 - ▶ **float**: floating point or real numbers

Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers
 - ▶ **float**: floating point or real numbers
 - ▶ **string**: sequence of characters

Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers
 - ▶ **float**: floating point or real numbers
 - ▶ **string**: sequence of characters
 - ▶ **list**: a sequence of items

Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers
 - ▶ **float**: floating point or real numbers
 - ▶ **string**: sequence of characters
 - ▶ **list**: a sequence of items
e.g. [3, 1, 4, 5, 9] or ['violet', 'purple', 'indigo']

Week 2: variables, data types, more on loops & range()






- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers
 - ▶ **float**: floating point or real numbers
 - ▶ **string**: sequence of characters
 - ▶ **list**: a sequence of items
e.g. [3, 1, 4, 5, 9] or ['violet', 'purple', 'indigo']
 - ▶ **class variables**: for complex objects, like turtles.

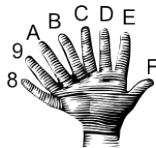
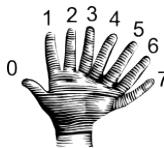
Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
 - ▶ **int**: integer or whole numbers
 - ▶ **float**: floating point or real numbers
 - ▶ **string**: sequence of characters
 - ▶ **list**: a sequence of items
e.g. [3, 1, 4, 5, 9] or ['violet', 'purple', 'indigo']
 - ▶ **class variables**: for complex objects, like turtles.
- More on loops & ranges:






```
1 #Predict what will be printed:
2
3 for num in [2,4,6,8,10]:
4     print(num)
5
6 sum = 0
7 for x in range(0,12,2):
8     print(x)
9     sum = sum + x
10
11 print(sum)
12
13 for c in "ABCD":
14     print(c)
```

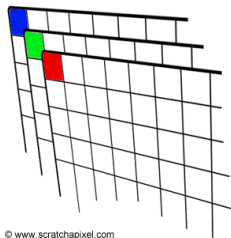
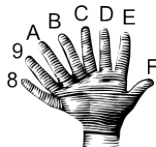
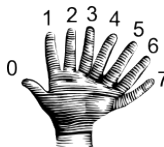
Week 3: colors, hex, slices, numpy & images

Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	








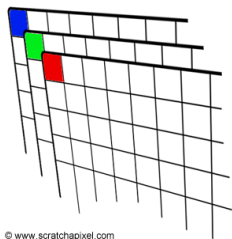
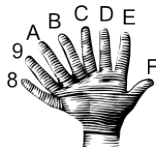
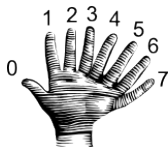
Week 3: colors, hex, slices, numpy & images

Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



Week 3: colors, hex, slices, numpy & images

Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



```
>>> a[0,3:5]
array([3,4])
```

```
>>> a[4:,4:]
array([[44, 45],
       [54, 55]])
```

```
>>> a[:,2]
array([2,12,22,32,42,52])
```

```
>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Week 4: design problem (cropping images) & decisions



Week 4: design problem (cropping images) & decisions



- First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right* (“bounding box”)

Week 4: design problem (cropping images) & decisions



- First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right* (“bounding box”)
- Next: write pseudocode.
 - ① Import numpy and pyplot.
 - ② Ask user for file names and dimensions for cropping.
 - ③ Save input file to an array.
 - ④ Copy the cropped portion to a new array.
 - ⑤ Save the new array to the output file.

Week 4: design problem (cropping images) & decisions



- First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right* (“bounding box”)
- Next: write pseudocode.
 - ① Import numpy and pyplot.
 - ② Ask user for file names and dimensions for cropping.
 - ③ Save input file to an array.
 - ④ Copy the cropped portion to a new array.
 - ⑤ Save the new array to the output file.
- Next: translate to Python.

Week 4: design problem (cropping images) & decisions

```
yearBorn = int(input('Enter year born: '))
if yearBorn < 1946:
    print("Greatest Generation")
elif yearBorn <= 1964:
    print("Baby Boomer")
elif yearBorn <= 1984:
    print("Generation X")
elif yearBorn <= 2004:
    print("Millennial")
else:
    print("TBD")

x = int(input('Enter number: '))
if x % 2 == 0:
    print('Even number')
else:
    print('Odd number')
```

Week 5: logical operators, truth tables & logical circuits

```
origin = "Indian Ocean"
winds = 100
if (winds > 74):
    print("Major storm, called a ", end="")
    if origin == "Indian Ocean" or origin == "South Pacific":
        print("cyclone.")
    elif origin == "North Pacific":
        print("typhoon.")
    else:
        print("hurricane.")

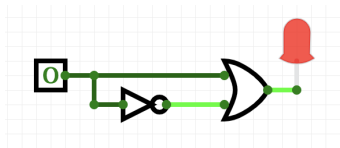
visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
    (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

Week 5: logical operators, truth tables & logical circuits

```
origin = "Indian Ocean"
winds = 100
if (winds > 74):
    print("Major storm, called a ", end="")
    if origin == "Indian Ocean" or origin == "South Pacific":
        print("cyclone.")
    elif origin == "North Pacific":
        print("typhoon.")
    else:
        print("hurricane.")

visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
    (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

in1		in2	returns:
False	and	False	False
False	and	True	False
True	and	False	False
True	and	True	True



Week 6: structured data, pandas, & more design

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City,....
All population figures are consistent with present-day boundaries.....
First census after the consolidation of the five boroughs.....
.....
.....
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island>Total
1698,4937,2017,,127,7881
1771,21883,3623,,2847,28423
1790,33131,4548,6159,1181,3827,49447
1800,40515,5740,6642,1755,4543,79215
1810,46373,6303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,3045,3023,7082,242278
1840,312710,47613,34480,5344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32903,23593,25492,1174779
1870,942282,419901,45468,37393,33029,1478183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,132899,20507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284183,2018356,448942,732018,116531,3620348
1930,1867312,2560451,1079129,1265208,150346,6305446
1940,1889924,2698285,1297634,1394711,174441,7454395
1950,1940101,2738275,1550849,1452277,191555,7893257
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7077439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332450,443728,8008278
2010,1648473,2504760,2230722,1385108,448730,8175133
2015,1644518,2636738,2339155,1455444,474558,8550405
```

nycHistPop.csv

In Lab 6

Week 6: structured data, pandas, & more design

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
Source: https://en.wikipedia.org/wiki/Demographics\_of\_New\_York\_City,....
All population figures are consistent with present-day boundaries.....
First census after the consolidation of the five boroughs.....
.....
.....
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,127,7881
1771,21883,3623,,2847,28423
1790,35131,4548,6159,1181,3827,49447
1800,40515,5740,6642,1755,4543,79215
1810,46373,6303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,8048,3023,7082,242278
1840,312710,47613,34480,5344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32903,23593,25492,1174779
1870,942282,419801,45648,37393,33829,1478183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1650093,1166582,115899,20507,67021,3437202
1910,2331542,1634351,284041,430989,85969,4766883
1920,2284183,2018356,448942,732018,116531,3420348
1930,1867312,2560451,1079129,1265298,159346,6905446
1940,1889924,2698285,1297634,1394711,174441,7454395
1950,1940101,2738275,1550849,1452277,191555,7893257
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7077439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332450,443728,8008278
2010,1648473,2504760,2230728,1385108,448738,8175133
2015,1644518,2636738,2339150,1455444,474558,8550405
```

nycHistPop.csv

In Lab 6

Week 6: structured data, pandas, & more design

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
pop = pd.read_csv('nycHistPop.csv', skiprows=5)
```

```
Source: https://en.wikipedia.org/wiki/Demographics\_of\_New\_York\_City,....
All population figures are consistent with present-day boundaries.....
First census after the consolidation of the five boroughs.....
.....
.....
Year,Manhattan, Brooklyn, Queens, Bronx, Staten Island, Total
1698,4937,2017,,127,7881
1771,21883,3623,,2847,28423
1790,20131,4548,6159,1181,3827,49447
1800,40515,5740,6642,1755,4543,79215
1810,96373,8303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,8048,3023,7082,242278
1840,312710,47613,14480,5344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32903,23593,25492,1174779
1870,942282,419801,45648,37393,33829,1478183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1650093,1166582,152899,200507,67021,3437202
1910,2331542,1634351,284041,430989,85969,4766883
1920,2284183,2018356,448942,732018,116531,3620348
1930,3867312,2560451,1079129,1265298,159346,6950446
1940,4889924,2698285,1297634,1394711,174441,7454395
1950,1960101,2738275,1550849,1452277,191559,7893257
1960,4698281,2627319,1809578,1424815,221991,7781984
1970,5339233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332450,443728,8008278
2010,1648473,2568760,2230720,1385108,448730,8175133
2015,1644518,2636738,2339150,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

Week 6: structured data, pandas, & more design

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
pop = pd.read_csv('nycHistPop.csv', skiprows=5)
```

```
Source: https://en.wikipedia.org/wiki/Demographics\_of\_New\_York\_City,.....
All population figures are consistent with present-day boundaries.....
First census after the consolidation of the five boroughs.....
```

```
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island>Total
1698,4937,2017,,127,7881
1773,21883,3623,,2847,28423
1790,35131,4548,6159,1181,3827,49447
1800,40515,5740,6642,1755,4543,79215
1810,46373,6303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,8048,3023,7082,242278
1840,312710,47613,14480,5344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32903,23593,25492,1174779
1870,942282,419801,45648,37393,33029,1478183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1650093,1166582,152899,200507,67021,3437202
1910,2331542,1634351,284041,430989,85969,4766883
1920,2284183,2018356,448942,732018,116531,5420348
1930,1867312,2580451,1079129,1265258,159346,6505446
1940,1889924,2698285,1297634,1394711,174441,7454395
1950,1960101,2738275,1550849,1452277,191555,7893257
1960,1698281,2627319,1809578,1424815,221991,7781986
1970,1539233,2402012,1986473,1471701,295443,7094862
1980,1428285,2230936,1891325,1168972,352121,7077439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332450,443728,8008278
2010,1648473,2504760,2230722,1385108,448730,8175133
2015,1644518,2636738,2339150,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

```
pop.plot(x="Year")
plt.show()
```

Week 6: structured data, pandas, & more design

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
pop = pd.read_csv('nycHistPop.csv', skiprows=5)
```

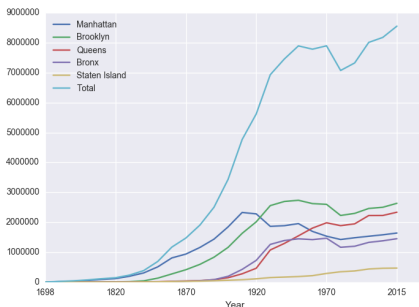
```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City,....
All population figures are consistent with present-day boundaries.....
First census after the consolidation of the five boroughs.....
.....
```

```
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,727,7481
1773,21863,3623,,2847,28423
1790,33131,4548,6159,1181,3827,49447
1800,40515,5740,6442,1755,4543,79215
1810,46373,6303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,8048,3023,7082,242278
1840,312710,47613,14480,5344,10965,393114
1850,515547,138882,18593,8032,15561,696115
1860,813649,279122,32903,23593,25492,1174779
1870,942282,419901,45468,37393,33529,1478183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51493,2507414
1900,1650093,1146582,152899,20507,67021,3437202
1910,2331542,1634351,284041,430989,85949,4766883
1920,2284193,2018356,448942,732018,114631,5420348
1930,1867312,2560451,1079125,1565269,159346,6950446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738275,1550849,1452277,191559,7893257
1960,1698281,2627319,1809578,1424815,221991,7781986
1970,1539233,2402012,1986473,1471701,295443,7094862
1980,1428285,2230936,1891325,1148972,352121,7077439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332450,443728,8008278
2010,1648473,2504769,2230722,1385108,448730,8175133
2015,1644518,2636735,2339150,1455444,474558,8550405
```

nycHistPop.csv

In Lab 6

```
pop.plot(x="Year")
plt.show()
```



Week 7: functions

- Functions are a way to break code into pieces, that can be easily reused.

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.

Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`

Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:

Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis: Example: `print("Hello", "World")`

Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis: Example: `print("Hello", "World")`
- Can write, or **define** your own functions,

Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis: Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked or called.

Week 8: function parameters, github

- Functions can have **input parameters**.

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: ' ))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: ' ))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

Week 8: function parameters, github

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: ' ))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: ' ))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

Week 8: function parameters, github

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: ' ))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: ' ))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.

Week 8: function parameters, github

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: ' ))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: ' ))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**

Week 8: function parameters, github

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: ' ))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: ' ))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**
- Functions can also **return values** to where it was called.

Week 8: function parameters, github

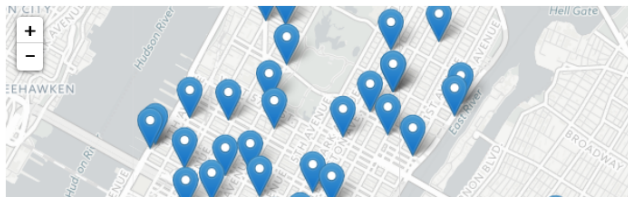
```
def totalWithTax(food,tip):  
    total = 0  
    tax = 0.0875  
    total = food + food * tax  
    total = total + tip  
    return(total)  
  
lunch = float(input('Enter lunch total: '))  
lTip = float(input('Enter lunch tip: ' ))  
lTotal = totalWithTax(lunch, lTip)  
print('Lunch total is', lTotal)  
  
dinner= float(input('Enter dinner total: '))  
dTip = float(input('Enter dinner tip: ' ))  
dTotal = totalWithTax(dinner, dTip)  
print('Dinner total is', dTotal)
```

Formal Parameters

Actual Parameters

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**.
- Functions can also **return values** to where it was called.

Week 9: top-down design, folium, loops, and random()



```
def main():  
    dataF = getData()  
    latColName, lonColName = getColumnNames()  
    lat, lon = getLocale()  
    cityMap = folium.Map(location = [lat,lon], tiles = 'cartodbpositron', zoom_start=11)  
    dotAllPoints(cityMap, dataF, latColName, lonColName)  
    markAndFindClosest(cityMap, dataF, latColName, lonColName, lat, lon)  
    writeMap(cityMap)
```

Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.

```
import turtle
import random

trex = turtle.Turtle()
trex.speed(10)

for i in range(100):
    trex.forward(10)
    a = random.randrange(0,360,90)
    trex.right(a)
```

Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.
- Very useful for checking user input for correctness.

```
import turtle
import random

trex = turtle.Turtle()
trex.speed(10)

for i in range(100):
    trex.forward(10)
    a = random.randrange(0,360,90)
    trex.right(a)
```

Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

```
import turtle
import random
```

```
trey = turtle.Turtle()
trey.speed(10)
```

```
for i in range(100):
    trey.forward(10)
    a = random.randrange(0,360,90)
    trey.right(a)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.
- Very useful for checking user input for correctness.
- Python's built-in random package has useful methods for generating random whole numbers and real numbers.

Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

```
import turtle
import random
```

```
trey = turtle.Turtle()
trey.speed(10)
```

```
for i in range(100):
    trey.forward(10)
    a = random.randrange(0,360,90)
    trey.right(a)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.
- Very useful for checking user input for correctness.
- Python's built-in random package has useful methods for generating random whole numbers and real numbers.
- To use, must include:
`import random.`

Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

```
import turtle
import random
```

```
trey = turtle.Turtle()
trey.speed(10)
```

```
for i in range(100):
    trey.forward(10)
    a = random.randrange(0,360,90)
    trey.right(a)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.
- Very useful for checking user input for correctness.
- Python's built-in random package has useful methods for generating random whole numbers and real numbers.
- To use, must include:
`import random.`
- The max design pattern provides a template for finding maximum value from a list.

Python & Circuits Review: 10 Weeks in 10 Minutes



- Input/Output (I/O): `input()` and `print()`;
pandas for CSV files
- Types:
 - ▶ Primitive: `int`, `float`, `bool`, `string`;
 - ▶ Container: lists (but not dictionaries/hashtes or tuples)
- Objects: turtles (used but did not design our own)
- Loops: definite & indefinite
- Conditionals: `if-elif-else`
- Logical Expressions & Circuits
- Functions: parameters & returns
- Packages:
 - ▶ Built-in: `turtle`, `math`, `random`
 - ▶ Popular: `numpy`, `matplotlib`, `pandas`, `folium`

Lecture Quiz

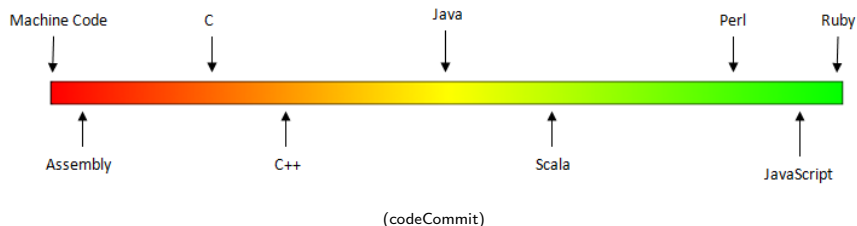
- Log-in to Gradescope
- Find LECTURE 11 Quiz
- Take the quiz
- **You have 3 minutes**

Today's Topics



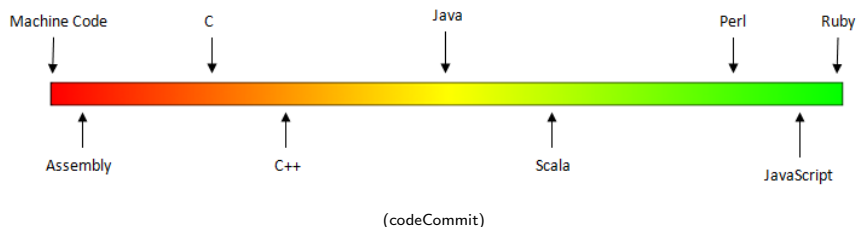
- Design Patterns: Searching
- Python Recap
- **Machine Language**
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

Low-Level vs. High-Level Languages



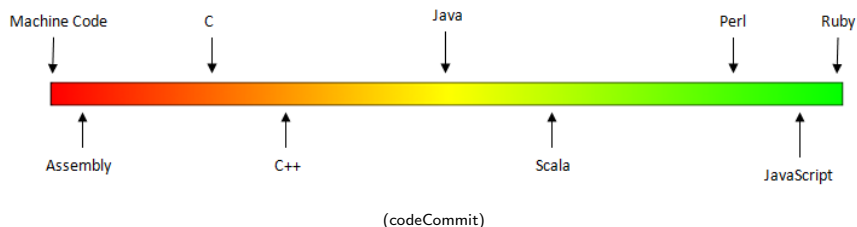
- Can view programming languages on a continuum.

Low-Level vs. High-Level Languages



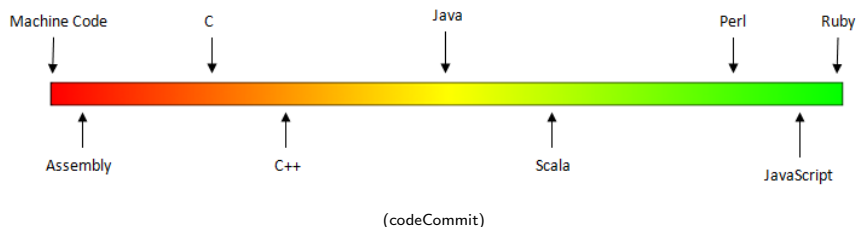
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

Low-Level vs. High-Level Languages



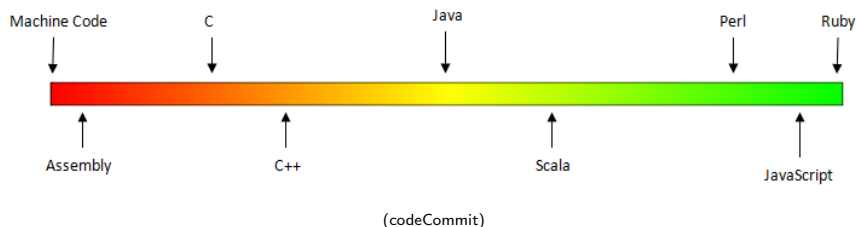
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

Low-Level vs. High-Level Languages



- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
- Those that have strong abstraction (allow programming paradigms independent of the machine details, such as complex variables, functions and looping that do not translate directly into machine code) are called **high-level languages**.

Low-Level vs. High-Level Languages



- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
- Those that have strong abstraction (allow programming paradigms independent of the machine details, such as complex variables, functions and looping that do not translate directly into machine code) are called **high-level languages**.
- Some languages, like C, are in between— allowing both low level access and high level data structures.

Processing



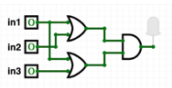
Dies ist ein Blindtext. An ihm lässt sich vieles über die Schrift ablesen, in der er gesetzt ist. Auf den ersten Blick wird der Grauwert der Schriftfläche sichtbar. Dann kann man prüfen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt. Dies ist ein Blindtext. An ihm lässt sich



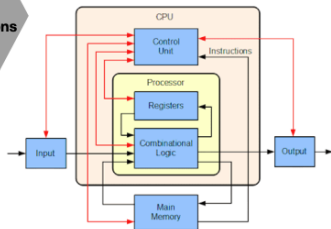
Data & Instructions



Data & Instructions

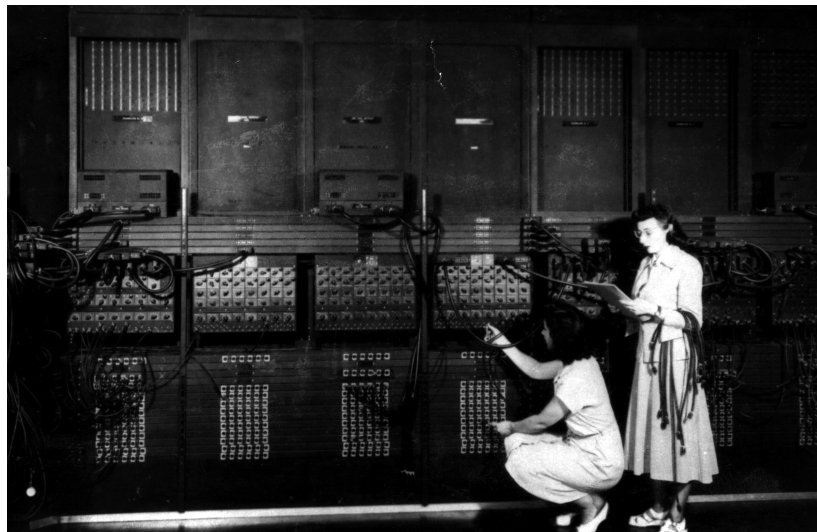


Circuits (switches)
On/Off 1/0 Logic
Billions of switches/bits



```
def totalWithTax(food,tip):  
    total = 0  
    tax = 0.0875  
    total = food + food * tax  
    total = total + tip  
    return(total)
```

Machine Language



(Ruth Gordon & Ester Gerston programming the ENIAC, UPenn)

Machine Language

```
1 FOX 12:01a 23- 1
A 002000 C2 30 REP #$30
A 002002 18 CLC
A 002003 F8 SED
A 002004 A9 34 12 LDA #$1234
A 002007 69 21 43 ADC #$4321
A 00200A 8F 03 7F 01 STA $017F03
A 00200E D8 CLD
A 00200F E2 30 SEP #$30
A 002011 00 BRK
A 2012

r
PB PC NUmxDIzC .A .X .Y SP DP DB
; 00 E012 00110000 0000 0000 0002 CFFF 0000 00
g 2000

BREAK

PB PC NUmxDIzC .A .X .Y SP DP DB
; 00 2013 00110000 5555 0000 0002 CFFF 0000 00
m 7f03 7f03
>007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00:UU.....
█
```

(wiki)

Machine Language

- We will be writing programs in a simplified machine language, WeMIPS.

```
002000 c2 30 REP #K30
002002 16 CLC
002004 F8 SED
002006 80 34 12 LSH #1234
002008 69 21 43 RLC #04321
00200A 0F 83 7F 81 STA #17F83
00200C 30 CLD
00200E E2 30 SEP #K30
002010 00 BRK
02012

P PC Mem012C A X Y SP BP
: 00 E012 00110000 0000 0000 0002 C7FF 0000 00
$ 2000

BREAK

P PC Mem012C A X Y SP BP
: 00 2013 00110000 5555 0000 0002 C7FF 0000 00
n 1183 7183
$0FF05 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(wiki)

Machine Language



```
002000 c2 30      REP #K30
002002 18          CLC
002004 f8          SED
002006 80 34 12    LSW #1234
002008 69 21 43    RLC #04321
00200a 0f 83 7f 81  STW #017f83
00200c 30          CLD
00200e e7 30      SEP #K30
002010 00          BRK
002012

r
r0 PC Mem012C  A  X  Y  SP  DP  EB
: 00 2012 00110000 0000 0000 0002 C7FF 0000 00
$ 2000

BREAK
r0 PC Mem012C  A  X  Y  SP  DP  EB
: 00 2013 00110000 5555 0000 0002 C7FF 0000 00
n 1183 7183
$0FFED 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(wiki)

- We will be writing programs in a simplified machine language, WeMIPS.
- It is based on a reduced instruction set computer (RISC) design, originally developed by the MIPS Computer Systems.

Machine Language



```
002000 C2 30 REP #430
002002 18 CLC
002004 F9 32 SET
002006 69 21 43 LSR #4324
002008 0F 83 7F 81 STW #017F83
00200E 30 CLD
002010 E7 30 SEP #430
002011 00 BRK
02012

PC Mem012C A X Y SP BP EB
: 00 2012 00110000 0000 0000 0002 C7FF 0000 00
$ 2000

BREAK

PC Mem012C A X Y SP BP EB
: 00 2013 00110000 5555 0000 0002 C7FF 0000 00
n 1183 7183
$00F0F5 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(wiki)

- We will be writing programs in a simplified machine language, WeMIPS.
- It is based on a reduced instruction set computer (RISC) design, originally developed by the MIPS Computer Systems.
- Due to its small set of commands, processors can be designed to run those commands very efficiently.
- More in future architecture classes....

"Hello World!" in Simplified Machine Language

Line: 3 Got

Show/Hide Demos

[User Guide](#) | [Unit Tests](#) | [Docs](#)

Addition Doubler Stav Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

```
1 # Store 'Hello world!' at the top of the stack
2 ADDI $sp, $sp, -13
3 ADDI $t0, $zero, 72 # H
4 SB $t0, 0($sp)
5 ADDI $t0, $zero, 101 # e
6 SB $t0, 1($sp)
7 ADDI $t0, $zero, 108 # l
8 SB $t0, 2($sp)
9 ADDI $t0, $zero, 108 # l
10 SB $t0, 3($sp)
11 ADDI $t0, $zero, 111 # o
12 SB $t0, 4($sp)
13 ADDI $t0, $zero, 32 # (space)
14 SB $t0, 5($sp)
15 ADDI $t0, $zero, 119 # w
16 SB $t0, 6($sp)
17 ADDI $t0, $zero, 111 # o
18 SB $t0, 7($sp)
19 ADDI $t0, $zero, 114 # r
20 SB $t0, 8($sp)
21 ADDI $t0, $zero, 108 # l
22 SB $t0, 9($sp)
23 ADDI $t0, $zero, 100 # d
24 SB $t0, 10($sp)
25 ADDI $t0, $zero, 33 # !
26 SB $t0, 11($sp)
27 ADDI $t0, $zero, 0 # (null)
28 SB $t0, 12($sp)
29
30 ADDI $v0, $zero, 4 # 4 is for print string
31 ADDI $a0, $sp, 0
32 syscall # print to the log
```

Step Run Enable auto switching

S T A V Stack Log

s0:	10
s1:	9
s2:	9
s3:	22
s4:	696
s5:	976
s6:	927
s7:	418

(WeMIPS)

WeMIPS

Line 3 `cat`

Show/Hide Demos

Addition Doubler `Stop` `Looper` `Stack Test` `Hello World`

Code Gen Save String `Interactive` `Binary2 Decimal` `Decimal2 Binary`

Debug

```
1 # Store 'hello world!' at the top of the stack
2 ADDI $a0, $zero, 32 # $0
3 SD $a0, 0($0)
4 ADDI $d0, $zero, 191 # w
5 SD $d0, 1($0)
6 ADDI $d0, $zero, 108 # l
7 SD $d0, 2($0)
8 ADDI $d0, $zero, 109 # l
9 SD $d0, 3($0)
10 ADDI $d0, $zero, 111 # o
11 SD $d0, 4($0)
12 ADDI $d0, $zero, 32 # (space)
13 SD $d0, 5($0)
14 ADDI $d0, $zero, 113 # w
15 SD $d0, 6($0)
16 ADDI $d0, $zero, 114 # a
17 SD $d0, 7($0)
18 ADDI $d0, $zero, 114 # a
19 SD $d0, 8($0)
20 ADDI $d0, $zero, 108 # l
21 SD $d0, 9($0)
22 ADDI $d0, $zero, 108 # l
23 SD $d0, 10($0)
24 ADDI $d0, $zero, 33 # !
25 SD $d0, 11($0)
26 ADDI $d0, $zero, 0 # (null)
27 SD $d0, 12($0)
28 #
29 ADDI $v0, $zero, 6 # $4 in for print string
30 ADDI $a0, $0, 0 # print to the log
31 syscall
```

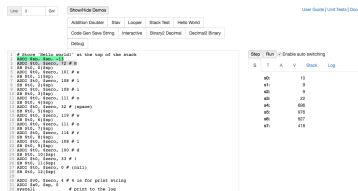
User Guide | Unit Tests | Docs

Step `Run` `Enable auto switching`

S	T	A	V	Stack	Log
				a0:	10
				a1:	9
				a2:	9
				a3:	22
				a4:	695
				a5:	976
				a6:	927
				a7:	418

(Demo with WeMIPS)

MIPS Commands



The screenshot shows a MIPS simulator window with several tabs: "Code Gen", "Data String", "Interactions", "Event", "Display", "Decimal", and "Hexadecimal Binary". The main window displays assembly code with line numbers 1 through 33. The code includes instructions like `addi $t0, $zero, 10`, `addi $t1, $zero, 100`, `addi $t2, $zero, 1000`, `addi $t3, $zero, 10000`, `addi $t4, $zero, 100000`, `addi $t5, $zero, 1000000`, `addi $t6, $zero, 10000000`, `addi $t7, $zero, 100000000`, `addi $t8, $zero, 1000000000`, `addi $t9, $zero, 10000000000`, `addi $t10, $zero, 100000000000`, `addi $t11, $zero, 1000000000000`, `addi $t12, $zero, 10000000000000`, `addi $t13, $zero, 100000000000000`, `addi $t14, $zero, 1000000000000000`, `addi $t15, $zero, 10000000000000000`, `addi $t16, $zero, 100000000000000000`, `addi $t17, $zero, 1000000000000000000`, `addi $t18, $zero, 10000000000000000000`, `addi $t19, $zero, 100000000000000000000`, `addi $t20, $zero, 1000000000000000000000`, `addi $t21, $zero, 10000000000000000000000`, `addi $t22, $zero, 100000000000000000000000`, `addi $t23, $zero, 1000000000000000000000000`, `addi $t24, $zero, 10000000000000000000000000`, `addi $t25, $zero, 100000000000000000000000000`, `addi $t26, $zero, 1000000000000000000000000000`, `addi $t27, $zero, 10000000000000000000000000000`, `addi $t28, $zero, 100000000000000000000000000000`, `addi $t29, $zero, 1000000000000000000000000000000`, `addi $t30, $zero, 10000000000000000000000000000000`, `addi $t31, $zero, 100000000000000000000000000000000`, `addi $t32, $zero, 1000000000000000000000000000000000`, `addi $t33, $zero, 10000000000000000000000000000000000`. The register file on the right shows registers \$0 through \$31 with their current values.

```
1 # MIPS "Hello world" on the top of the stack
2 addi $t0, $zero, 10
3 addi $t1, $zero, 100
4 addi $t2, $zero, 1000
5 addi $t3, $zero, 10000
6 addi $t4, $zero, 100000
7 addi $t5, $zero, 1000000
8 addi $t6, $zero, 10000000
9 addi $t7, $zero, 100000000
10 addi $t8, $zero, 1000000000
11 addi $t9, $zero, 10000000000
12 addi $t10, $zero, 100000000000
13 addi $t11, $zero, 1000000000000
14 addi $t12, $zero, 10000000000000
15 addi $t13, $zero, 100000000000000
16 addi $t14, $zero, 1000000000000000
17 addi $t15, $zero, 10000000000000000
18 addi $t16, $zero, 100000000000000000
19 addi $t17, $zero, 1000000000000000000
20 addi $t18, $zero, 10000000000000000000
21 addi $t19, $zero, 100000000000000000000
22 addi $t20, $zero, 1000000000000000000000
23 addi $t21, $zero, 10000000000000000000000
24 addi $t22, $zero, 100000000000000000000000
25 addi $t23, $zero, 1000000000000000000000000
26 addi $t24, $zero, 10000000000000000000000000
27 addi $t25, $zero, 100000000000000000000000000
28 addi $t26, $zero, 1000000000000000000000000000
29 addi $t27, $zero, 10000000000000000000000000000
30 addi $t28, $zero, 100000000000000000000000000000
31 addi $t29, $zero, 1000000000000000000000000000000
32 addi $t30, $zero, 10000000000000000000000000000000
33 # print to the joy
```

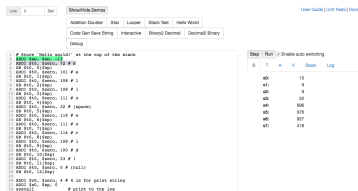
- **Registers:** locations for storing information that can be quickly accessed.

MIPS Commands

The screenshot shows a MIPS simulator interface. The main window displays assembly code with line numbers 1 through 33. The code includes comments and instructions such as `ADDI $s0, $zero, 10`, `ADDI $s1, $zero, 100`, `ADDI $t0, $zero, 1000`, `ADDI $t1, $zero, 10000`, `ADDI $s0, $s0, 10`, `ADDI $s1, $s1, 100`, `ADDI $t0, $t0, 1000`, `ADDI $t1, $t1, 10000`, `ADDI $s0, $s0, 100`, `ADDI $s1, $s1, 1000`, `ADDI $t0, $t0, 10000`, `ADDI $t1, $t1, 100000`, `ADDI $s0, $s0, 1000`, `ADDI $s1, $s1, 10000`, `ADDI $t0, $t0, 100000`, `ADDI $t1, $t1, 1000000`, and `syscall`. The register window on the right shows the state of registers \$0 through \$31, with \$0 at 0, \$1 at 10, \$2 at 100, \$3 at 1000, \$4 at 10000, \$5 at 100000, \$6 at 1000000, \$7 at 10000000, and registers \$8 through \$31 at 0.

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...

MIPS Commands



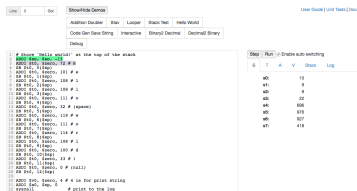
The screenshot shows a MIPS simulator interface. At the top, there are tabs for 'Code Gen View String', 'Interactions', 'Event/Decom', and 'Decom/Binary'. Below these is a 'Setting' section. The main area displays assembly code with line numbers 1 through 33. The code includes instructions like 'li \$s0, 1000000', 'add \$s1, \$s0, 100 # 1', 'add \$s2, \$s0, 100 # 2', and 'add \$s3, \$s0, 100 # 3'. On the right side, there is a 'Register' window with a table showing the values of registers \$0 through \$7.

```
1 # Shows "Hello world" on the top of the stack
2 addi $s0, $zero, 1000000
3 li $s1, 1000000
4 add $s2, $s0, 100 # 1
5 add $s3, $s0, 100 # 2
6 add $s4, $s0, 100 # 3
7 add $s5, $s0, 100 # 4
8 add $s6, $s0, 100 # 5
9 add $s7, $s0, 100 # 6
10 add $s8, $s0, 100 # 7
11 add $s9, $s0, 100 # 8
12 add $s10, $s0, 100 # 9
13 add $s11, $s0, 100 # 10
14 add $s12, $s0, 100 # 11
15 add $s13, $s0, 100 # 12
16 add $s14, $s0, 100 # 13
17 add $s15, $s0, 100 # 14
18 add $s16, $s0, 100 # 15
19 add $s17, $s0, 100 # 16
20 add $s18, $s0, 100 # 17
21 add $s19, $s0, 100 # 18
22 add $s20, $s0, 100 # 19
23 add $s21, $s0, 100 # 20
24 add $s22, $s0, 100 # 21
25 add $s23, $s0, 100 # 22
26 add $s24, $s0, 100 # 23
27 add $s25, $s0, 100 # 24
28 add $s26, $s0, 100 # 25
29 add $s27, $s0, 100 # 26
30 add $s28, $s0, 100 # 27
31 add $s29, $s0, 100 # 28
32 add $s30, $s0, 100 # 29
33 # print to the top
```

Register	Value
\$0	0
\$1	10
\$2	100
\$3	200
\$4	300
\$5	400
\$6	500
\$7	600

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:

MIPS Commands



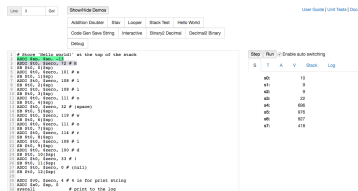
The screenshot shows a MIPS simulator window with a menu bar (File, Edit, View, Show/Hide Digits, Address Decoder, Bus, Loader, Stack Test, Help Window) and a toolbar (Code Gen Save String, Interactions, Element Decoder, Decimal Binary). The main area displays assembly code with comments. A 'Registers' window on the right shows a table of registers with their values.

```
1 # Shows "Hello world" on the top of the stack
2 ADDI $a0, $zero, 10 #R
3 SW $a0, 0($zero) #R
4 ADDI $a0, $zero, 10 #R
5 SW $a0, 4($zero) #R
6 ADDI $a0, $zero, 100 #R
7 SW $a0, 8($zero) #R
8 ADDI $a0, $zero, 100 #R
9 SW $a0, 12($zero) #R
10 ADDI $a0, $zero, 10 #R
11 SW $a0, 16($zero) #R
12 ADDI $a0, $zero, 10 #R
13 SW $a0, 20($zero) #R
14 ADDI $a0, $zero, 100 #R
15 SW $a0, 24($zero) #R
16 ADDI $a0, $zero, 100 #R
17 SW $a0, 28($zero) #R
18 ADDI $a0, $zero, 10 #R
19 SW $a0, 32($zero) #R
20 ADDI $a0, $zero, 0 #R (null)
21 SW $a0, 36($zero) #R
22 ADDI $a0, $zero, 4 #R in the print string
23 ADDI $a0, $zero, 0 #R in the jmp
24 syscall
```

Register	Value
\$0	0
\$1	10
\$2	0
\$3	100
\$4	10
\$5	100
\$6	10
\$7	100

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:
add \$s1, \$s2, \$s3

MIPS Commands



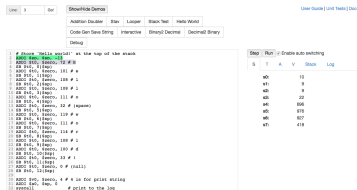
The screenshot shows a MIPS simulator interface. At the top, there are buttons for 'Show/Hide Demos', 'Address Decoder', 'CPU', 'Loader', 'Stack Test', and 'Hello World'. Below these are buttons for 'Code Gen: Save String', 'Interactions', 'Eternal Decimals', and 'Decimal Binary'. A 'Disasm' button is also present. The main area displays assembly code with comments. The register window on the right shows the state of registers \$0 through \$31.

```
# Shows "Hello world" on the top of the stack
1 addi $a0, $zero, 10
2 addi $a1, $zero, 11
3 addi $a2, $zero, 12
4 addi $a3, $zero, 13
5 addi $a4, $zero, 14
6 addi $a5, $zero, 15
7 addi $a6, $zero, 16
8 addi $a7, $zero, 17
9 addi $a8, $zero, 18
10 addi $a9, $zero, 19
11 addi $a10, $zero, 20
12 addi $a11, $zero, 21
13 addi $a12, $zero, 22
14 addi $a13, $zero, 23
15 addi $a14, $zero, 24
16 addi $a15, $zero, 25
17 addi $a16, $zero, 26
18 addi $a17, $zero, 27
19 addi $a18, $zero, 28
20 addi $a19, $zero, 29
21 addi $a20, $zero, 30
22 addi $a21, $zero, 31
23 addi $a22, $zero, 32
24 addi $a23, $zero, 33
25 addi $a24, $zero, 34
26 addi $a25, $zero, 35
27 addi $a26, $zero, 36
28 addi $a27, $zero, 37
29 addi $a28, $zero, 38
30 addi $a29, $zero, 39
31 addi $a30, $zero, 40
32 addi $a31, $zero, 41
```

Hex	Val	Enable auto-scrolling
\$0	0	
\$1	10	
\$2	11	
\$3	12	
\$4	13	
\$5	14	
\$6	15	
\$7	16	
\$8	17	
\$9	18	
\$10	19	
\$11	20	
\$12	21	
\$13	22	
\$14	23	
\$15	24	
\$16	25	
\$17	26	
\$18	27	
\$19	28	
\$20	29	
\$21	30	
\$22	31	
\$23	32	
\$24	33	
\$25	34	
\$26	35	
\$27	36	
\$28	37	
\$29	38	
\$30	39	
\$31	40	

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:
add \$s1, \$s2, \$s3 (Basic form: OP rd, rs, rt)
- **I Instructions:** instructions that also use intermediate values.

MIPS Commands



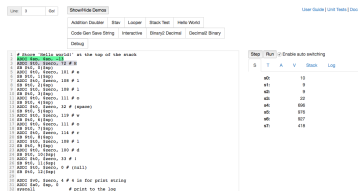
The screenshot shows a MIPS simulator window. The main area displays assembly code with comments. The register window on the right shows the state of registers \$0 through \$31.

```
# Shows "Hello world" on the top of the stack
addi $0, $0, 0
addi $1, $0, 10
addi $2, $0, 20
addi $3, $0, 30
addi $4, $0, 40
addi $5, $0, 50
addi $6, $0, 60
addi $7, $0, 70
addi $8, $0, 80
addi $9, $0, 90
addi $10, $0, 100
addi $11, $0, 110
addi $12, $0, 120
addi $13, $0, 130
addi $14, $0, 140
addi $15, $0, 150
addi $16, $0, 160
addi $17, $0, 170
addi $18, $0, 180
addi $19, $0, 190
addi $20, $0, 200
addi $21, $0, 210
addi $22, $0, 220
addi $23, $0, 230
addi $24, $0, 240
addi $25, $0, 250
addi $26, $0, 260
addi $27, $0, 270
addi $28, $0, 280
addi $29, $0, 290
addi $30, $0, 300
addi $31, $0, 310
```

Hex	Dec	Enable auto-scrolling
\$0	0	
\$1	10	
\$2	20	
\$3	30	
\$4	40	
\$5	50	
\$6	60	
\$7	70	
\$8	80	
\$9	90	
\$10	100	
\$11	110	
\$12	120	
\$13	130	
\$14	140	
\$15	150	
\$16	160	
\$17	170	
\$18	180	
\$19	190	
\$20	200	
\$21	210	
\$22	220	
\$23	230	
\$24	240	
\$25	250	
\$26	260	
\$27	270	
\$28	280	
\$29	290	
\$30	300	
\$31	310	

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:
add \$s1, \$s2, \$s3 (Basic form: OP rd, rs, rt)
- **I Instructions:** instructions that also use intermediate values.
addi \$s1, \$s2, 100

MIPS Commands



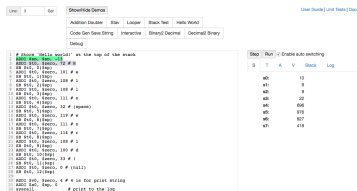
The screenshot shows a MIPS simulator window. The main area displays assembly code with comments. The register window on the right shows the state of registers \$0 through \$31.

```
# Shows "Hello world" as the top of the stack
ADDI $0, $0, 0
ADDI $1, $0, 10
ADDI $2, $0, 100
ADDI $3, $0, 1000
ADDI $4, $0, 10000
ADDI $5, $0, 100000
ADDI $6, $0, 1000000
ADDI $7, $0, 10000000
ADDI $8, $0, 100000000
ADDI $9, $0, 1000000000
ADDI $10, $0, 10000000000
ADDI $11, $0, 100000000000
ADDI $12, $0, 1000000000000
ADDI $13, $0, 10000000000000
ADDI $14, $0, 100000000000000
ADDI $15, $0, 1000000000000000
ADDI $16, $0, 10000000000000000
ADDI $17, $0, 100000000000000000
ADDI $18, $0, 1000000000000000000
ADDI $19, $0, 10000000000000000000
ADDI $20, $0, 100000000000000000000
ADDI $21, $0, 1000000000000000000000
ADDI $22, $0, 10000000000000000000000
ADDI $23, $0, 100000000000000000000000
ADDI $24, $0, 1000000000000000000000000
ADDI $25, $0, 10000000000000000000000000
ADDI $26, $0, 100000000000000000000000000
ADDI $27, $0, 1000000000000000000000000000
ADDI $28, $0, 10000000000000000000000000000
ADDI $29, $0, 100000000000000000000000000000
ADDI $30, $0, 1000000000000000000000000000000
ADDI $31, $0, 10000000000000000000000000000000
```

Hex	Dec	String
\$0	0	
\$1	10	
\$2	100	
\$3	1000	
\$4	10000	
\$5	100000	
\$6	1000000	
\$7	10000000	
\$8	100000000	
\$9	1000000000	
\$10	10000000000	
\$11	100000000000	
\$12	1000000000000	
\$13	10000000000000	
\$14	100000000000000	
\$15	1000000000000000	
\$16	10000000000000000	
\$17	100000000000000000	
\$18	1000000000000000000	
\$19	10000000000000000000	
\$20	100000000000000000000	
\$21	1000000000000000000000	
\$22	10000000000000000000000	
\$23	100000000000000000000000	
\$24	1000000000000000000000000	
\$25	10000000000000000000000000	
\$26	100000000000000000000000000	
\$27	1000000000000000000000000000	
\$28	10000000000000000000000000000	
\$29	100000000000000000000000000000	
\$30	1000000000000000000000000000000	
\$31	10000000000000000000000000000000	

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:
add \$s1, \$s2, \$s3 (Basic form: OP rd, rs, rt)
- **I Instructions:** instructions that also use intermediate values.
addi \$s1, \$s2, 100 (Basic form: OP rd, rs, imm)
- **J Instructions:** instructions that jump to another memory location.

MIPS Commands



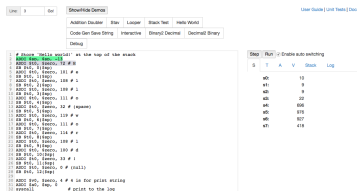
The screenshot shows a MIPS simulator window. The main area contains assembly code with comments. The register file on the right shows the state of registers \$0 through \$31.

```
# Shows "Hello world" as the top of the stack
addi $0, $0, 0
addi $1, $0, 10
addi $2, $0, 100 # s
addi $3, $0, 1000 # d
addi $4, $0, 10000 # f
addi $5, $0, 100000 # g
addi $6, $0, 1000000 # h
addi $7, $0, 10000000 # i
addi $8, $0, 100000000 # j
addi $9, $0, 1000000000 # k
addi $10, $0, 10000000000 # l
addi $11, $0, 100000000000 # m
addi $12, $0, 1000000000000 # n
addi $13, $0, 10000000000000 # o
addi $14, $0, 100000000000000 # p
addi $15, $0, 1000000000000000 # q
addi $16, $0, 10000000000000000 # r
addi $17, $0, 100000000000000000 # s
addi $18, $0, 1000000000000000000 # t
addi $19, $0, 10000000000000000000 # u
addi $20, $0, 100000000000000000000 # v
addi $21, $0, 1000000000000000000000 # w
addi $22, $0, 10000000000000000000000 # x
addi $23, $0, 100000000000000000000000 # y
addi $24, $0, 1000000000000000000000000 # z
addi $25, $0, 10000000000000000000000000 # aa
addi $26, $0, 100000000000000000000000000 # ab
addi $27, $0, 1000000000000000000000000000 # ac
addi $28, $0, 10000000000000000000000000000 # ad
addi $29, $0, 100000000000000000000000000000 # ae
addi $30, $0, 1000000000000000000000000000000 # af
addi $31, $0, 10000000000000000000000000000000 # ag
```

Hex	Dec	String
\$0	0	
\$1	10	
\$2	100	
\$3	1000	
\$4	10000	
\$5	100000	
\$6	1000000	
\$7	10000000	
\$8	100000000	
\$9	1000000000	
\$10	10000000000	
\$11	100000000000	
\$12	1000000000000	
\$13	10000000000000	
\$14	100000000000000	
\$15	1000000000000000	
\$16	10000000000000000	
\$17	100000000000000000	
\$18	1000000000000000000	
\$19	10000000000000000000	
\$20	100000000000000000000	
\$21	1000000000000000000000	
\$22	10000000000000000000000	
\$23	100000000000000000000000	
\$24	1000000000000000000000000	
\$25	10000000000000000000000000	
\$26	100000000000000000000000000	
\$27	1000000000000000000000000000	
\$28	10000000000000000000000000000	
\$29	100000000000000000000000000000	
\$30	1000000000000000000000000000000	
\$31	10000000000000000000000000000000	

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:
add \$s1, \$s2, \$s3 (Basic form: OP rd, rs, rt)
- **I Instructions:** instructions that also use intermediate values.
addi \$s1, \$s2, 100 (Basic form: OP rd, rs, imm)
- **J Instructions:** instructions that jump to another memory location.
j done

MIPS Commands



The screenshot shows a MIPS simulator window. The main area displays assembly code with comments. The register window on the right shows the state of registers \$0 through \$31.

```
# Shows "Hello world" as the top of the stack
addi $0, $0, 0
addi $1, $0, 10
addi $2, $0, 100 # 2
addi $3, $0, 1000 # 1
addi $4, $0, 10000 # 1
addi $5, $0, 100000 # 1
addi $6, $0, 1000000 # 1
addi $7, $0, 10000000 # 1
addi $8, $0, 100000000 # 1
addi $9, $0, 1000000000 # 1
addi $10, $0, 10000000000 # 1
addi $11, $0, 100000000000 # 1
addi $12, $0, 1000000000000 # 1
addi $13, $0, 10000000000000 # 1
addi $14, $0, 100000000000000 # 1
addi $15, $0, 1000000000000000 # 1
addi $16, $0, 10000000000000000 # 1
addi $17, $0, 100000000000000000 # 1
addi $18, $0, 1000000000000000000 # 1
addi $19, $0, 10000000000000000000 # 1
addi $20, $0, 100000000000000000000 # 1
addi $21, $0, 1000000000000000000000 # 1
addi $22, $0, 10000000000000000000000 # 1
addi $23, $0, 100000000000000000000000 # 1
addi $24, $0, 1000000000000000000000000 # 1
addi $25, $0, 10000000000000000000000000 # 1
addi $26, $0, 100000000000000000000000000 # 1
addi $27, $0, 1000000000000000000000000000 # 1
addi $28, $0, 10000000000000000000000000000 # 1
addi $29, $0, 100000000000000000000000000000 # 1
addi $30, $0, 1000000000000000000000000000000 # 1
addi $31, $0, 10000000000000000000000000000000 # 1
```

Hex	Dec	String
\$0	0	
\$1	10	
\$2	100	
\$3	1000	
\$4	10000	
\$5	100000	
\$6	1000000	
\$7	10000000	
\$8	100000000	
\$9	1000000000	
\$10	10000000000	
\$11	100000000000	
\$12	1000000000000	
\$13	10000000000000	
\$14	100000000000000	
\$15	1000000000000000	
\$16	10000000000000000	
\$17	100000000000000000	
\$18	1000000000000000000	
\$19	10000000000000000000	
\$20	100000000000000000000	
\$21	1000000000000000000000	
\$22	10000000000000000000000	
\$23	100000000000000000000000	
\$24	1000000000000000000000000	
\$25	10000000000000000000000000	
\$26	100000000000000000000000000	
\$27	1000000000000000000000000000	
\$28	10000000000000000000000000000	
\$29	100000000000000000000000000000	
\$30	1000000000000000000000000000000	
\$31	10000000000000000000000000000000	

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:
add \$s1, \$s2, \$s3 (Basic form: OP rd, rs, rt)
- **I Instructions:** instructions that also use intermediate values.
addi \$s1, \$s2, 100 (Basic form: OP rd, rs, imm)
- **J Instructions:** instructions that jump to another memory location.
j done (Basic form: OP label)

Challenge:

Line: 3 Go!

Show/Hide Demos

[User Guide](#) | [Unit Tests](#) | [Docs](#)

Addition Doubler

Stav

Looper

Stack Test

Hello World

Code Gen Save String

Interactive

Binary2 Decimal

Decimal2 Binary

Debug

```
1 # Store 'Hello world!' at the top of the stack
2 ADDI $sp, $sp, -13
3 ADDI $t0, $zero, 72 # H
4 SB $t0, 0($sp)
5 ADDI $t0, $zero, 101 # e
6 SB $t0, 1($sp)
7 ADDI $t0, $zero, 108 # l
8 SB $t0, 2($sp)
9 ADDI $t0, $zero, 108 # l
10 SB $t0, 3($sp)
11 ADDI $t0, $zero, 111 # o
12 SB $t0, 4($sp)
13 ADDI $t0, $zero, 32 # (space)
14 SB $t0, 5($sp)
15 ADDI $t0, $zero, 119 # w
16 SB $t0, 6($sp)
17 ADDI $t0, $zero, 111 # o
18 SB $t0, 7($sp)
19 ADDI $t0, $zero, 114 # r
20 SB $t0, 8($sp)
21 ADDI $t0, $zero, 108 # l
22 SB $t0, 9($sp)
23 ADDI $t0, $zero, 100 # d
24 SB $t0, 10($sp)
25 ADDI $t0, $zero, 33 # !
26 SB $t0, 11($sp)
27 ADDI $t0, $zero, 0 # (null)
28 SB $t0, 12($sp)
29
30 ADDI $v0, $zero, 4 # 4 is for print string
31 ADDI $a0, $sp, 0
32 syscall # print to the log
```

Step Run Enable auto switching

S	T	A	V	Stack	Log
				s0:	10
				s1:	9
				s2:	9
				s3:	22
				s4:	696
				s5:	976
				s6:	927
				s7:	418

Write a program that prints out the alphabet: a b c d ... x y z

WeMIPS

Line Out Show/Hide Demos User Guide | Unit Tests | Docs

Addition Doubler Star Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

```
1 # Store 'hello world!' at the top of the stack
2 ADDI $a0, $zero, 32 # $0
3 SD $a0, 0($zero)
4 ADDI $d0, $zero, 191 # w
5 SD $d0, 1($zero)
6 ADDI $d0, $zero, 108 # l
7 SD $d0, 2($zero)
8 ADDI $d0, $zero, 109 # l
9 SD $d0, 3($zero)
10 ADDI $d0, $zero, 111 # o
11 SD $d0, 4($zero)
12 ADDI $d0, $zero, 32 # (space)
13 SD $d0, 5($zero)
14 ADDI $d0, $zero, 113 # w
15 SD $d0, 6($zero)
16 ADDI $d0, $zero, 114 # a
17 SD $d0, 7($zero)
18 ADDI $d0, $zero, 114 # a
19 SD $d0, 8($zero)
20 ADDI $d0, $zero, 108 # l
21 SD $d0, 9($zero)
22 ADDI $d0, $zero, 108 # l
23 SD $d0, 10($zero)
24 ADDI $d0, $zero, 33 # !
25 SD $d0, 11($zero)
26 ADDI $d0, $zero, 0 # (null)
27 SD $d0, 12($zero)
28
29 ADDI $v0, $zero, 6 # $4 in for print string
30 ADDI $a0, $zero, 0 # print to the log
31 syscall
```

Step Run Enable auto switching

S	T	A	V	Stack	Log
				a0:	10
				v0:	9
				a0:	9
				v0:	22
				a0:	905
				v0:	976
				a0:	927
				v0:	418

(Demo with WeMIPS)

Today's Topics



- Design Patterns: Searching
- Python Recap
- Machine Language
- **Machine Language: Jumps & Loops**
- Binary & Hex Arithmetic
- Final Exam: Format

Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.



The screenshot shows a debugger window with two panes. The left pane displays assembly code with instructions such as `movl $0, %eax`, `movl $1, %eax`, and `jmp $0x00000000, %eax`. The right pane shows the state of registers, including `%eax` containing the value `0x00000001`.

Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.



The screenshot shows a debugger window with two panes. The left pane displays assembly code with several lines of instructions, including `jmp` (jump) instructions. The right pane shows the corresponding machine code in hexadecimal and binary. The assembly code includes labels like `loop_start` and `loop_end`, and instructions such as `jmp loop_start` which create a loop.

Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:
 - ▶ **Unconditional:** `j Done` will jump to the address with label `Done`.



The screenshot shows a debugger window with two panes. The left pane displays assembly instructions with their corresponding memory addresses and hex values. The right pane shows the disassembled instructions with labels. A label 'Done' is visible at the beginning of a line, and a jump instruction 'j Done' is shown below it, illustrating the unconditional jump mechanism.

Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:
 - ▶ **Unconditional:** `j Done` will jump to the address with label `Done`.
 - ▶ **Branch if Equal:** `beq $s0 $s1 DoAgain` will jump to the address with label `DoAgain` if the registers `$s0` and `$s1` contain the same value.



```

# MIPS assembly code for a loop
# Example: A loop that prints 'Hello' 5 times
.data
    str: .asciz "Hello\n"
    count: .word 5
.text
    la $t0, count
loop_start:
    li $v0, 4
    la $a0, str
    syscall
    li $v0, 10
    syscall
    addi $t0, $t0, -1
    bne $t0, $zero, loop_start
    li $v0, 10
    syscall

```

Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:
 - ▶ **Unconditional:** `j Done` will jump to the address with label `Done`.
 - ▶ **Branch if Equal:** `beq $s0 $s1 DoAgain` will jump to the address with label `DoAgain` if the registers `$s0` and `$s1` contain the same value.
 - ▶ See reading for more variations.



```

# Example 1.1: A simple loop in assembly language
# The program prints the numbers 1 through 10.

    .text
    .globl main
    .type main, @function
main:
    li $v0, 1          # Print integer
    li $a0, 1          # Argument: 1
    syscall            # Print integer

    li $v0, 4          # Print string
    li $a0, "\n"       # Argument: newline
    syscall            # Print string

    li $v0, 1          # Print integer
    li $a0, 1          # Argument: 1
    syscall            # Print integer

    li $v0, 4          # Print string
    li $a0, "\n"       # Argument: newline
    syscall            # Print string

    li $v0, 1          # Print integer
    li $a0, 2          # Argument: 2
    syscall            # Print integer

    li $v0, 4          # Print string
    li $a0, "\n"       # Argument: newline
    syscall            # Print string

    li $v0, 1          # Print integer
    li $a0, 3          # Argument: 3
    syscall            # Print integer

    li $v0, 4          # Print string
    li $a0, "\n"       # Argument: newline
    syscall            # Print string

    li $v0, 1          # Print integer
    li $a0, 4          # Argument: 4
    syscall            # Print integer

    li $v0, 4          # Print string
    li $a0, "\n"       # Argument: newline
    syscall            # Print string

    li $v0, 1          # Print integer
    li $a0, 5          # Argument: 5
    syscall            # Print integer

    li $v0, 4          # Print string
    li $a0, "\n"       # Argument: newline
    syscall            # Print string

    li $v0, 1          # Print integer
    li $a0, 6          # Argument: 6
    syscall            # Print integer

    li $v0, 4          # Print string
    li $a0, "\n"       # Argument: newline
    syscall            # Print string

    li $v0, 1          # Print integer
    li $a0, 7          # Argument: 7
    syscall            # Print integer

    li $v0, 4          # Print string
    li $a0, "\n"       # Argument: newline
    syscall            # Print string

    li $v0, 1          # Print integer
    li $a0, 8          # Argument: 8
    syscall            # Print integer

    li $v0, 4          # Print string
    li $a0, "\n"       # Argument: newline
    syscall            # Print string

    li $v0, 1          # Print integer
    li $a0, 9          # Argument: 9
    syscall            # Print integer

    li $v0, 4          # Print string
    li $a0, "\n"       # Argument: newline
    syscall            # Print string

    li $v0, 1          # Print integer
    li $a0, 10         # Argument: 10
    syscall            # Print integer

    li $v0, 4          # Print string
    li $a0, "\n"       # Argument: newline
    syscall            # Print string

    li $v0, 10         # Exit
    syscall            # Exit

    .end main

```

Jump Demo

Line: 18

Go!

Show/Hide Demos

[User Guide](#) | [Unit Tests](#) | [Docs](#)

```
1
2 ADDI $sp, $sp, -27      # Set up stack
3 ADDI $s3, $zero, 1     # Store 1 in a register
4 ADDI $t0, $zero, 97    # Set $t0 at 97 (a)
5 ADDI $s2, $zero, 26    # Use to test when you reach 26
6 SETUP: SB $t0, 0($sp)  # Next letter in $t0
7 ADDI $sp, $sp, 1       # Increment the stack
8 SUB $s2, $s2, $s3      # Decrease the counter by 1
9 ADDI $t0, $t0, 1       # Increment the letter
10 BEQ $s2, $zero, DONE  # Jump to done if $s2 == 0
11 J SETUP                # Else, jump back to SETUP
12 DONE: ADDI $t0, $zero, 0 # Null (0) to terminate string
13 SB $t0, 0($sp)        # Add null to stack
14 ADDI $sp, $sp, -26    # Set up stack to print
15 ADDI $v0, $zero, 4    # 4 is for print string
16 ADDI $a0, $sp, 0      # Set $a0 to stack pointer
17 syscall              # Print to the log
```

(Demo
with
WeMIPS)

Step **Run** Enable auto switching

S T A V Stack Log

Clear Log

Emulation complete, returning to line 1

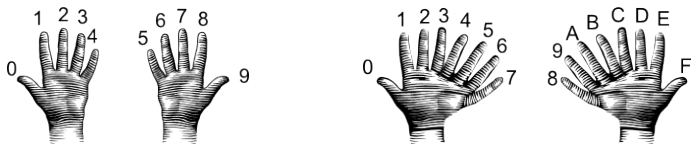
abcdefghijklmnopqrstuvwxyz

Today's Topics



- Design Patterns: Searching
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- **Binary & Hex Arithmetic**
- Final Exam: Format

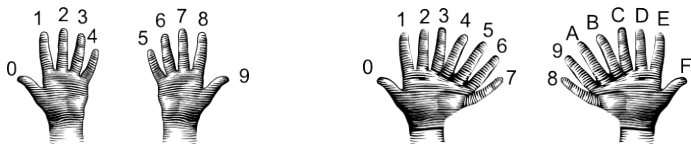
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - ▶ Convert first digit to decimal and multiple by 16.

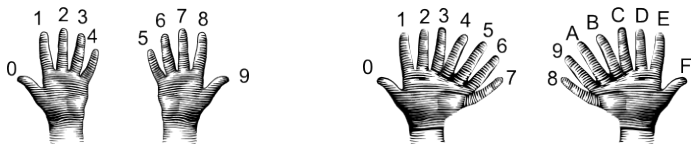
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - ▶ Convert first digit to decimal and multiple by 16.
 - ▶ Convert second digit to decimal and add to total.

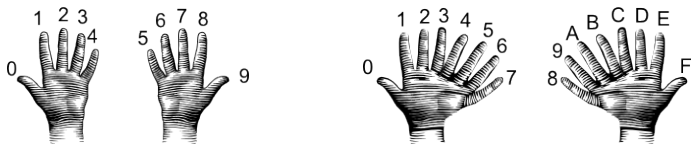
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - ▶ Convert first digit to decimal and multiple by 16.
 - ▶ Convert second digit to decimal and add to total.
 - ▶ Example: what is 2A as a decimal number?

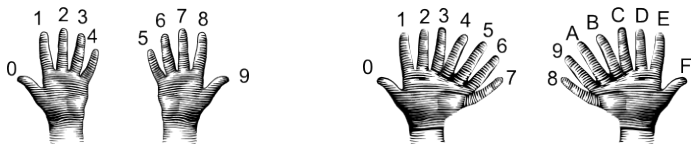
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - ▶ Convert first digit to decimal and multiple by 16.
 - ▶ Convert second digit to decimal and add to total.
 - ▶ Example: what is 2A as a decimal number?
2 in decimal is 2.

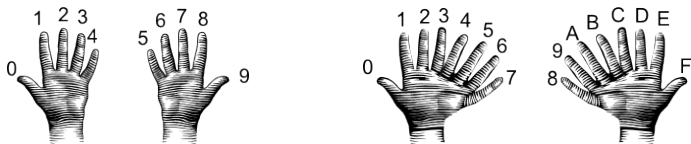
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - ▶ Convert first digit to decimal and multiple by 16.
 - ▶ Convert second digit to decimal and add to total.
 - ▶ Example: what is 2A as a decimal number?
2 in decimal is 2. 2×16 is 32.

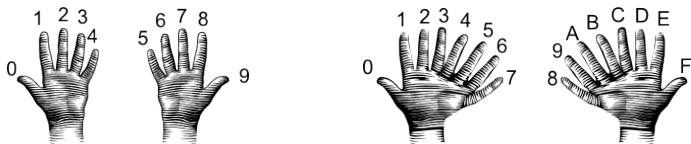
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - ▶ Convert first digit to decimal and multiple by 16.
 - ▶ Convert second digit to decimal and add to total.
 - ▶ Example: what is 2A as a decimal number?
2 in decimal is 2. 2×16 is 32.
A in decimal digits is 10.

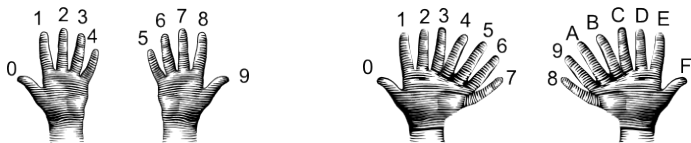
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - ▶ Convert first digit to decimal and multiple by 16.
 - ▶ Convert second digit to decimal and add to total.
 - ▶ Example: what is 2A as a decimal number?
 - 2 in decimal is 2. 2×16 is 32.
 - A in decimal digits is 10.
 - $32 + 10$ is 42.

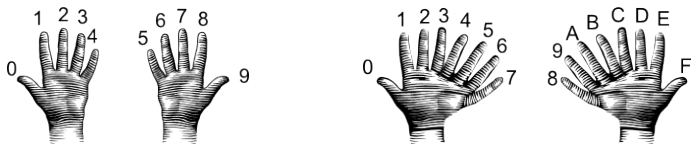
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - ▶ Convert first digit to decimal and multiple by 16.
 - ▶ Convert second digit to decimal and add to total.
 - ▶ Example: what is 2A as a decimal number?
 - 2 in decimal is 2. 2×16 is 32.
 - A in decimal digits is 10.
 - $32 + 10$ is 42.
 - Answer is 42.
 - ▶ Example: what is 99 as a decimal number?

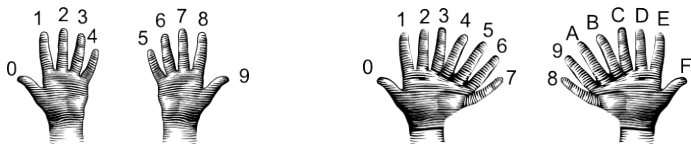
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - ▶ Convert first digit to decimal and multiple by 16.
 - ▶ Convert second digit to decimal and add to total.
 - ▶ Example: what is 2A as a decimal number?
2 in decimal is 2. 2×16 is 32.
A in decimal digits is 10.
 $32 + 10$ is 42.
Answer is 42.
 - ▶ Example: what is 99 as a decimal number?
9 in decimal is 9.

Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?

2 in decimal is 2. 2×16 is 32.

A in decimal digits is 10.

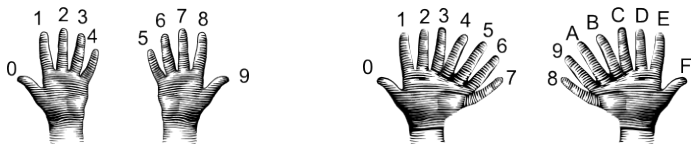
$32 + 10$ is 42.

Answer is 42.

- ▶ Example: what is 99 as a decimal number?

9 in decimal is 9. 9×16 is 144.

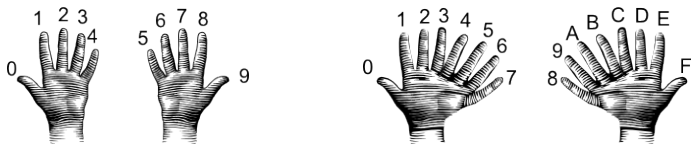
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - ▶ Convert first digit to decimal and multiple by 16.
 - ▶ Convert second digit to decimal and add to total.
 - ▶ Example: what is 2A as a decimal number?
 - 2 in decimal is 2. 2×16 is 32.
 - A in decimal digits is 10.
 - $32 + 10$ is 42.
 - Answer is 42.
 - ▶ Example: what is 99 as a decimal number?
 - 9 in decimal is 9. 9×16 is 144.
 - 9 in decimal digits is 9

Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?

2 in decimal is 2. 2×16 is 32.

A in decimal digits is 10.

$32 + 10$ is 42.

Answer is 42.

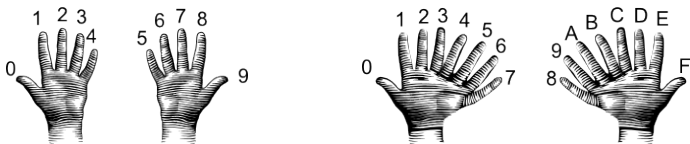
- ▶ Example: what is 99 as a decimal number?

9 in decimal is 9. 9×16 is 144.

9 in decimal digits is 9

$144 + 9$ is 153.

Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?

2 in decimal is 2. 2×16 is 32.

A in decimal digits is 10.

$32 + 10$ is 42.

Answer is 42.

- ▶ Example: what is 99 as a decimal number?

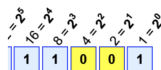
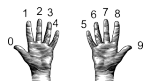
9 in decimal is 9. 9×16 is 144.

9 in decimal digits is 9

$144 + 9$ is 153.

Answer is 153.

Decimal to Binary: Converting Between Bases

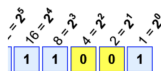


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.

Decimal to Binary: Converting Between Bases

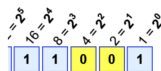
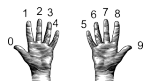


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.

Decimal to Binary: Converting Between Bases

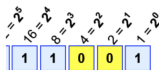


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.

Decimal to Binary: Converting Between Bases

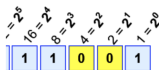


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.

Decimal to Binary: Converting Between Bases

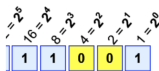


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.

Decimal to Binary: Converting Between Bases

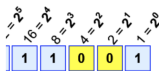


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.

Decimal to Binary: Converting Between Bases

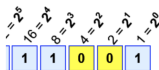


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.

Decimal to Binary: Converting Between Bases

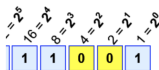


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.

Decimal to Binary: Converting Between Bases

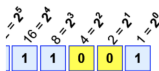


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

Decimal to Binary: Converting Between Bases



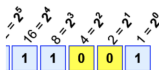
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2.

Decimal to Binary: Converting Between Bases



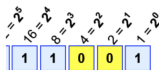
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1:

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

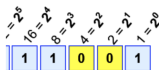
From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

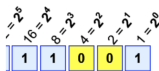
● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0:

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

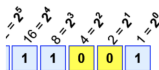
● From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From decimal to binary:

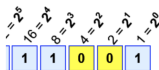
- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

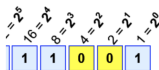
- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0:

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

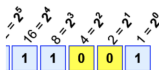
130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

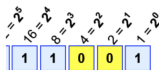
130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0:

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

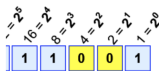
130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

Decimal to Binary: Converting Between Bases



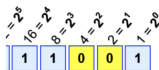
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2.

Decimal to Binary: Converting Between Bases



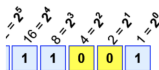
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0:

Decimal to Binary: Converting Between Bases



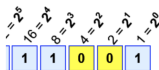
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...

Decimal to Binary: Converting Between Bases



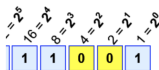
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...
2/4 is 0 remainder 2.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

2/8 is 0 rem 2. Next digit is 0: 10000...

2/4 is 0 remainder 2. Next digit is 0:

Decimal to Binary: Converting Between Bases



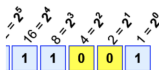
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...
2/4 is 0 remainder 2. Next digit is 0: 100000...

Decimal to Binary: Converting Between Bases



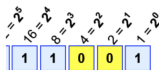
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...
2/4 is 0 remainder 2. Next digit is 0: 100000...
2/2 is 1 rem 0.

Decimal to Binary: Converting Between Bases



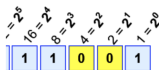
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...
2/4 is 0 remainder 2. Next digit is 0: 100000...
2/2 is 1 rem 0. Next digit is 1:

Decimal to Binary: Converting Between Bases



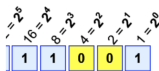
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...
2/4 is 0 remainder 2. Next digit is 0: 100000...
2/2 is 1 rem 0. Next digit is 1: 1000001...

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

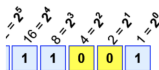
2/8 is 0 rem 2. Next digit is 0: 10000...

2/4 is 0 remainder 2. Next digit is 0: 100000...

2/2 is 1 rem 0. Next digit is 1: 1000001...

Adding the last remainder: 10000010

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient is the next digit.
- ▶ Divide remainder by 4 ($= 2^2$). Quotient is the next digit.
- ▶ Divide remainder by 2 ($= 2^1$). Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

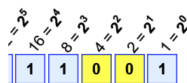
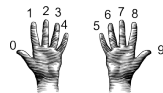
2/8 is 0 rem 2. Next digit is 0: 10000...

2/4 is 0 remainder 2. Next digit is 0: 100000...

2/2 is 1 rem 0. Next digit is 1: 1000001...

Adding the last remainder: 10000010

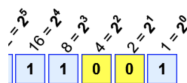
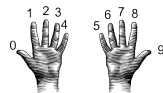
Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

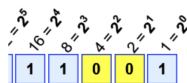
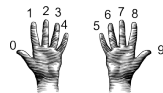
Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?
99/128 is 0 rem 99.

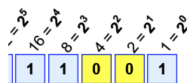
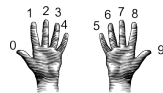
Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?
99/128 is 0 rem 99. First digit is 0:

Decimal to Binary: Converting Between Bases



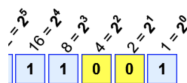
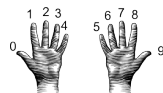
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

$99/128$ is 0 rem 99. First digit is 0: 0...

$99/64$ is 1 rem 35.

Decimal to Binary: Converting Between Bases



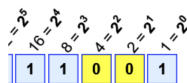
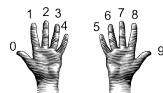
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

$99/128$ is 0 rem 99. First digit is 0: 0...

$99/64$ is 1 rem 35. Next digit is 1:

Decimal to Binary: Converting Between Bases



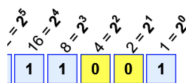
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

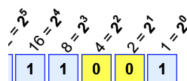
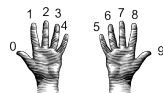
- Example: what is 99 in binary notation?

$99/128$ is 0 rem 99. First digit is 0: 0...

$99/64$ is 1 rem 35. Next digit is 1: 01...

$35/32$ is 1 rem 3.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

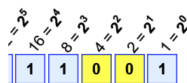
- Example: what is 99 in binary notation?

$99/128$ is 0 rem 99. First digit is 0: 0...

$99/64$ is 1 rem 35. Next digit is 1: 01...

$35/32$ is 1 rem 3. Next digit is 1:

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

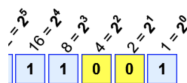
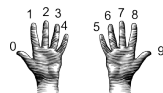
- Example: what is 99 in binary notation?

$99/128$ is 0 rem 99. First digit is 0: 0...

$99/64$ is 1 rem 35. Next digit is 1: 01...

$35/32$ is 1 rem 3. Next digit is 1: 011...

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

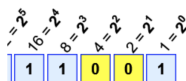
$99/128$ is 0 rem 99. First digit is 0: 0...

$99/64$ is 1 rem 35. Next digit is 1: 01...

$35/32$ is 1 rem 3. Next digit is 1: 011...

$3/16$ is 0 rem 3.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

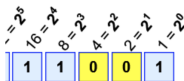
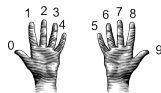
$99/128$ is 0 rem 99. First digit is 0: 0...

$99/64$ is 1 rem 35. Next digit is 1: 01...

$35/32$ is 1 rem 3. Next digit is 1: 011...

$3/16$ is 0 rem 3. Next digit is 0:

Decimal to Binary: Converting Between Bases

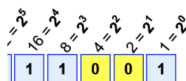
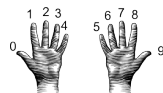


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

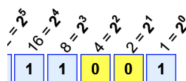
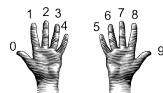
99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

$99/128$ is 0 rem 99. First digit is 0: 0...

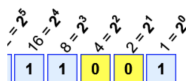
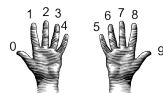
$99/64$ is 1 rem 35. Next digit is 1: 01...

$35/32$ is 1 rem 3. Next digit is 1: 011...

$3/16$ is 0 rem 3. Next digit is 0: 0110...

$3/8$ is 0 rem 3. Next digit is 0:

Decimal to Binary: Converting Between Bases

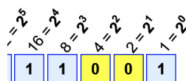
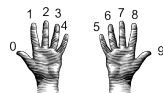


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3. Next digit is 0: 01100...

Decimal to Binary: Converting Between Bases

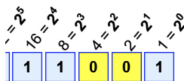
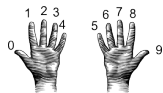


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3. Next digit is 0: 01100...
3/4 is 0 remainder 3.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

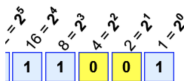
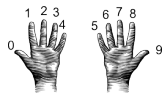
35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3. Next digit is 0:

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

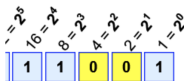
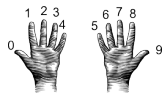
35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3. Next digit is 0: 011000...

Decimal to Binary: Converting Between Bases

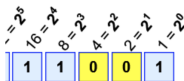
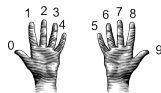


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3. Next digit is 0: 01100...
3/4 is 0 remainder 3. Next digit is 0: 011000...
3/2 is 1 rem 1.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

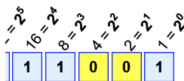
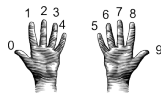
3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3. Next digit is 0: 011000...

3/2 is 1 rem 1. Next digit is 1:

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

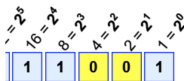
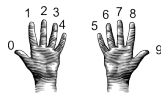
3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3. Next digit is 0: 011000...

3/2 is 1 rem 1. Next digit is 1: 0110001...

Decimal to Binary: Converting Between Bases

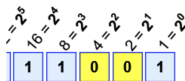
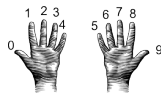


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...
99/64 is 1 rem 35. Next digit is 1: 01...
35/32 is 1 rem 3. Next digit is 1: 011...
3/16 is 0 rem 3. Next digit is 0: 0110...
3/8 is 0 rem 3. Next digit is 0: 01100...
3/4 is 0 remainder 3. Next digit is 0: 011000...
3/2 is 1 rem 1. Next digit is 1: 0110001...
Adding the last remainder: 01100011

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

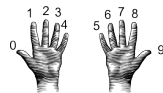
3/4 is 0 remainder 3. Next digit is 0: 011000...

3/2 is 1 rem 1. Next digit is 1: 0110001...

Adding the last remainder: 01100011

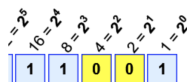
Answer is 1100011.

Binary to Decimal: Converting Between Bases



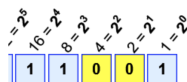
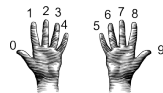
● From binary to decimal:

- ▶ Set sum = last digit.



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

Binary to Decimal: Converting Between Bases

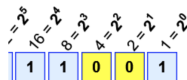
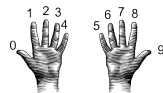


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.

Binary to Decimal: Converting Between Bases

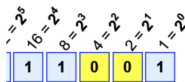
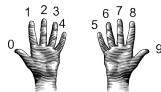


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.

Binary to Decimal: Converting Between Bases

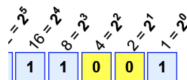
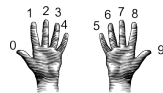


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.

Binary to Decimal: Converting Between Bases

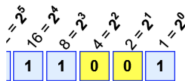
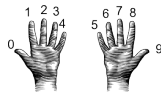


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.

Binary to Decimal: Converting Between Bases

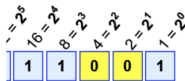
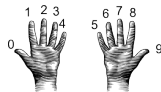


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.

Binary to Decimal: Converting Between Bases

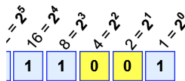
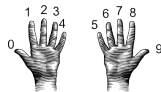


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.

Binary to Decimal: Converting Between Bases

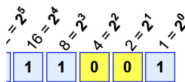
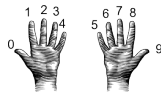


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.

Binary to Decimal: Converting Between Bases

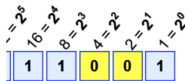
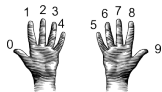


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.

Binary to Decimal: Converting Between Bases



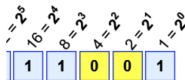
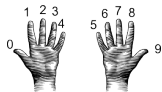
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with:

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

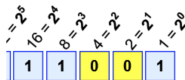
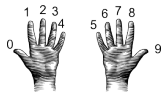
● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with: 1

$0 \times 2 = 0$. Add 0 to sum:

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

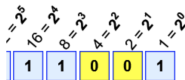
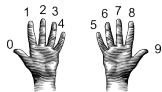
● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with: 1

$0 \times 2 = 0$. Add 0 to sum: 1

Binary to Decimal: Converting Between Bases



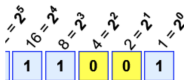
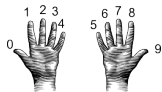
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with: 1
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum:

Binary to Decimal: Converting Between Bases



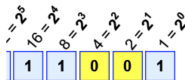
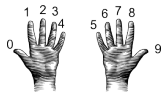
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with: 1
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum: 5

Binary to Decimal: Converting Between Bases



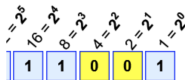
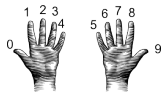
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with: 1
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum: 5
 $1 \times 8 = 8$. Add 8 to sum:

Binary to Decimal: Converting Between Bases



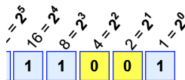
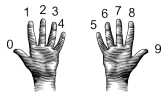
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with:	1
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	13

Binary to Decimal: Converting Between Bases



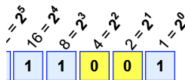
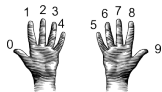
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

```
Sum starts with:           1
0*2 = 0.  Add 0 to sum:    1
1*4 = 4.  Add 4 to sum:    5
1*8 = 8.  Add 8 to sum:   13
1*16 = 16. Add 16 to sum:
```


Binary to Decimal: Converting Between Bases



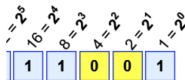
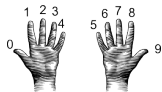
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with:	1
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	13
$1 \times 16 = 16$. Add 16 to sum:	29

Binary to Decimal: Converting Between Bases



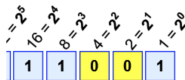
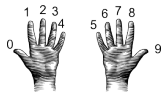
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with: 1
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum: 5
 $1 \times 8 = 8$. Add 8 to sum: 13
 $1 \times 16 = 16$. Add 16 to sum: 29
 $1 \times 32 = 32$. Add 32 to sum:

Binary to Decimal: Converting Between Bases



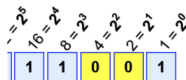
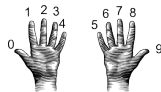
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with:	1
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	13
$1 \times 16 = 16$. Add 16 to sum:	29
$1 \times 32 = 32$. Add 32 to sum:	61

Binary to Decimal: Converting Between Bases



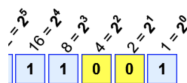
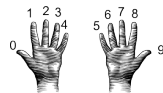
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.
- ▶ Multiply next digit by $4 = 2^2$. Add to sum.
- ▶ Multiply next digit by $8 = 2^3$. Add to sum.
- ▶ Multiply next digit by $16 = 2^4$. Add to sum.
- ▶ Multiply next digit by $32 = 2^5$. Add to sum.
- ▶ Multiply next digit by $64 = 2^6$. Add to sum.
- ▶ Multiply next digit by $128 = 2^7$. Add to sum.
- ▶ Sum is the decimal number.
- ▶ Example: What is 111101 in decimal?

Sum starts with:	1
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	13
$1 \times 16 = 16$. Add 16 to sum:	29
$1 \times 32 = 32$. Add 32 to sum:	61

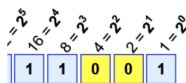
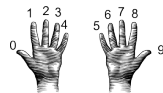
Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?
Sum starts with:

Binary to Decimal: Converting Between Bases



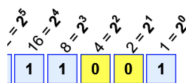
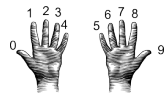
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum:

Binary to Decimal: Converting Between Bases



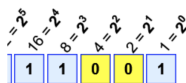
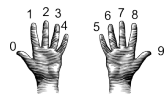
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 * 2 = 0$. Add 0 to sum: 0

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

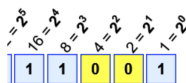
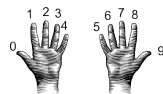
- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum:

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

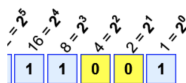
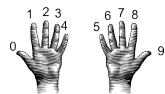
- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

Binary to Decimal: Converting Between Bases

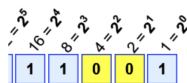
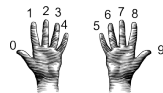


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0
 $0 \times 2 = 0$. Add 0 to sum: 0
 $1 \times 4 = 4$. Add 4 to sum: 4
 $0 \times 8 = 0$. Add 0 to sum:

Binary to Decimal: Converting Between Bases

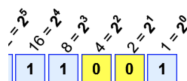
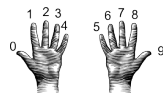


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0
 $0 \times 2 = 0$. Add 0 to sum: 0
 $1 \times 4 = 4$. Add 4 to sum: 4
 $0 \times 8 = 0$. Add 0 to sum: 4

Binary to Decimal: Converting Between Bases

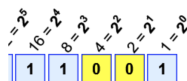
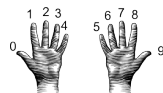


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0
 $0 \times 2 = 0$. Add 0 to sum: 0
 $1 \times 4 = 4$. Add 4 to sum: 4
 $0 \times 8 = 0$. Add 0 to sum: 4
 $0 \times 16 = 0$. Add 0 to sum: 4

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

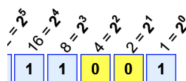
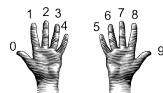
$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. Add 0 to sum: 4

$0 \times 16 = 0$. Add 0 to sum: 4

Binary to Decimal: Converting Between Bases

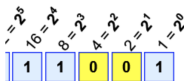
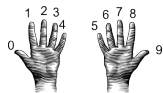


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0
 $0 \times 2 = 0$. Add 0 to sum: 0
 $1 \times 4 = 4$. Add 4 to sum: 4
 $0 \times 8 = 0$. Add 0 to sum: 4
 $0 \times 16 = 0$. Add 0 to sum: 4
 $1 \times 32 = 32$. Add 32 to sum:

Binary to Decimal: Converting Between Bases

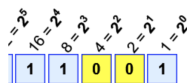
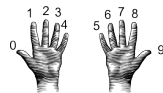


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0$. Add 0 to sum:	0
$1 \times 4 = 4$. Add 4 to sum:	4
$0 \times 8 = 0$. Add 0 to sum:	4
$0 \times 16 = 0$. Add 0 to sum:	4
$1 \times 32 = 32$. Add 32 to sum:	36

Binary to Decimal: Converting Between Bases

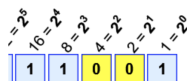
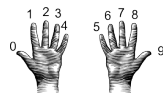


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0$. Add 0 to sum:	0
$1 \times 4 = 4$. Add 4 to sum:	4
$0 \times 8 = 0$. Add 0 to sum:	4
$0 \times 16 = 0$. Add 0 to sum:	4
$1 \times 32 = 32$. Add 32 to sum:	36
$0 \times 64 = 0$. Add 0 to sum:	

Binary to Decimal: Converting Between Bases

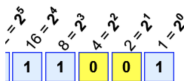
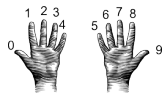


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0$. Add 0 to sum:	0
$1 \times 4 = 4$. Add 4 to sum:	4
$0 \times 8 = 0$. Add 0 to sum:	4
$0 \times 16 = 0$. Add 0 to sum:	4
$1 \times 32 = 32$. Add 32 to sum:	36
$0 \times 64 = 0$. Add 0 to sum:	36

Binary to Decimal: Converting Between Bases

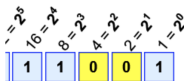
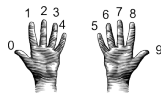


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0$. Add 0 to sum:	0
$1 \times 4 = 4$. Add 4 to sum:	4
$0 \times 8 = 0$. Add 0 to sum:	4
$0 \times 16 = 0$. Add 0 to sum:	4
$1 \times 32 = 32$. Add 32 to sum:	36
$0 \times 64 = 0$. Add 0 to sum:	36
$1 \times 128 = 0$. Add 128 to sum:	

Binary to Decimal: Converting Between Bases

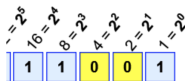
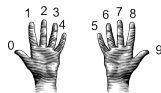


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0$. Add 0 to sum:	0
$1 \times 4 = 4$. Add 4 to sum:	4
$0 \times 8 = 0$. Add 0 to sum:	4
$0 \times 16 = 0$. Add 0 to sum:	4
$1 \times 32 = 32$. Add 32 to sum:	36
$0 \times 64 = 0$. Add 0 to sum:	36
$1 \times 128 = 128$. Add 128 to sum:	164

Binary to Decimal: Converting Between Bases



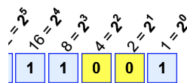
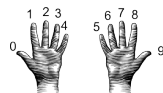
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0$. Add 0 to sum:	0
$1 \times 4 = 4$. Add 4 to sum:	4
$0 \times 8 = 0$. Add 0 to sum:	4
$0 \times 16 = 0$. Add 0 to sum:	4
$1 \times 32 = 32$. Add 32 to sum:	36
$0 \times 64 = 0$. Add 0 to sum:	36
$1 \times 128 = 128$. Add 128 to sum:	164

The answer is 164.

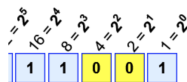
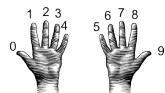
Design Challenge: Incrementers



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Simplest arithmetic: add one (“increment”) a variable.

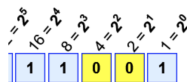
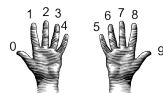
Design Challenge: Incrementers



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

Design Challenge: Incrementers

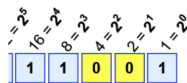
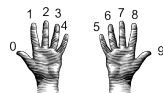


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```

Design Challenge: Incrementers

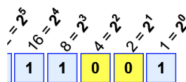
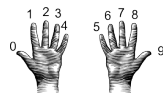


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```
- Challenge: Write an algorithm for incrementing numbers expressed as words.

Design Challenge: Incrementers

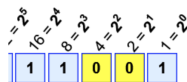
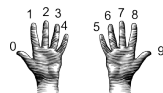


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```
- Challenge: Write an algorithm for incrementing numbers expressed as words.
Example: "forty one" → "forty two"

Design Challenge: Incrementers

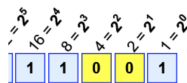


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```
- **Challenge:** Write an algorithm for incrementing numbers expressed as words.
Example: "forty one" → "forty two"
Hint: Convert to numbers, increment, and convert back to strings.

Design Challenge: Incrementers

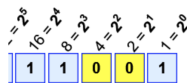
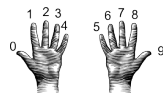


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```
- Challenge: Write an algorithm for incrementing numbers expressed as words.
Example: "forty one" → "forty two"
Hint: Convert to numbers, increment, and convert back to strings.
- Challenge: Write an algorithm for incrementing binary numbers.

Design Challenge: Incrementers



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```
- Challenge: Write an algorithm for incrementing numbers expressed as words.
Example: "forty one" → "forty two"
Hint: Convert to numbers, increment, and convert back to strings.
- Challenge: Write an algorithm for incrementing binary numbers.
Example: "1001" → "1010"

Recap



- Searching through data is a common task– built-in functions and standard design patterns for this.

Recap



- Searching through data is a common task– built-in functions and standard design patterns for this.
- Programming languages can be classified by the level of abstraction and direct access to data.

Today's Topics



- Design Patterns: Searching
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- **Final Exam: Format**

Final Overview: Administration

- The exam will be administered through Gradescope.

Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on during the time of the exam

Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on during the time of the exam
- There will be a different Gradescope Course called **CSci 127 Final Exam**

Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on during the time of the exam
- There will be a different Gradescope Course called **CSci 127 Final Exam**
- Prior to the exam you will be added to the final exam course for your exam version.

Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on during the time of the exam
- There will be a different Gradescope Course called **CSci 127 Final Exam**
- Prior to the exam you will be added to the final exam course for your exam version.
- The only assignment in that course will be your final exam.

Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on during the time of the exam
- There will be a different Gradescope Course called **CSci 127 Final Exam**
- Prior to the exam you will be added to the final exam course for your exam version.
- The only assignment in that course will be your final exam.
- The morning of the exam: log into Gradescope, find the **CSci 127 Final Exam** course and open the assignment.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Best if you design/write yours since excellent way to study.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Best if you design/write yours since excellent way to study.
 - ▶ Avoid scrambling through web searches and waste time during the exam.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Best if you design/write yours since excellent way to study.
 - ▶ Avoid scrambling through web searches and waste time during the exam.
- The exam format:

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Best if you design/write yours since excellent way to study.
 - ▶ Avoid scrambling through web searches and waste time during the exam.
- The exam format:
 - ▶ Like a long Lab Quiz, you scroll down to answer all questions.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Best if you design/write yours since excellent way to study.
 - ▶ Avoid scrambling through web searches and waste time during the exam.
- The exam format:
 - ▶ Like a long Lab Quiz, you scroll down to answer all questions.
 - ▶ Questions roughly correspond to the 10 parts from old exams, but will appear as a larger number of questions on Gradescope
 - ▶ Questions are variations on the programming assignments, lab exercises, and lecture design challenges.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Best if you design/write yours since excellent way to study.
 - ▶ Avoid scrambling through web searches and waste time during the exam.
- The exam format:
 - ▶ Like a long Lab Quiz, you scroll down to answer all questions.
 - ▶ Questions roughly correspond to the 10 parts from old exams, but will appear as a larger number of questions on Gradescope
 - ▶ Questions are variations on the programming assignments, lab exercises, and lecture design challenges.
- Past exams available on webpage (includes answer keys).

Exam Options

Exam Times:

FINAL EXAM, VERSION 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

19 December 2020

Exam Rules

- Write all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of one 4 1/2" x 6" piece of paper filled with notes, programs, etc.
- While taking the exam, you may have with you pens and pencils, and your work sheet.
- You may not use a calculator, cell-phone, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College respects acts of academic dishonesty (i.e., plagiarism, cheating on examinations, obtaining unfair advantage, and fabrication of records and official documents) as serious offenses against the values of educational integrity. The College is committed to upholding the CUNY Hunter College Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

Continuation page of exam of academic dishonesty will be reported to the Office of Institutional and Title IX Affairs.
Name:
Faculty:
Class:
Signature:

Exam Options

Exam Times:

- Default: Regular Time: Monday, 14 December, 9-11am.

FINAL EXAM, VERSION 3
CSI 127: Introduction to Computer Science
Hunter College, City University of New York

10 December 2020

Exam Rules

- Write all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of one 4 1/2" x 6" piece of paper filled with notes, programs, etc.
- While taking the exam, you may have with you pens and pencils, and your watch/clock.
- You may not use a calculator, cell/phone, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College respects acts of academic dishonesty (i.e., plagiarism, cheating on examinations, obtaining unfair advantage, and fabrication of records and official documents) as serious offenses against the values of educational integrity. The College is committed to upholding the CUNY Hunter College Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

Continuation page of exam of academic dishonesty will be reported to the Office of Academic and Professional Integrity.
Name:
Faculty:
Class:
Signature:

Exam Options

Exam Times:

- Default: Regular Time: Monday, 14 December, 9-11am.
- Alternate Time: Reading Day, Friday, 11 December, 8am-10am.

FISAL EXAM, VERSION 3
CSI112: Introduction to Computer Science
Hunter College, City University of New York

10 December 2020

Exam Rules

- Write all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of one 4 1/2" x 6" piece of paper filled with notes, programs, etc.
- While taking the exam, you may have with you pens and pencils, and your watch/clock.
- You may not use a calculator, cell-phone, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College respects acts of academic dishonesty (i.e., plagiarism, cheating on examinations, obtaining unfair advantage, and fabrication of records and official documents) as serious offenses against the values of educational integrity. The College is committed to upholding the CUNY Policy on Academic Integrity and will pursue acts of academic dishonesty according to the Hunter College Academic Integrity Procedures.

1. A handwritten copy of your act of academic dishonesty will be required to file. Name of student and ID# (with ID#):
Name:
ID#:
Event:
Signature:

Exam Options

Exam Times:

- **Default: Regular Time: Monday, 14 December, 9-11am.**
- **Alternate Time: Reading Day, Friday, 11 December, 8am-10am.**
- **Accessibility Testing: If you have not done so already, email me no later than 7 December.**

FISAL EXAM, VERSION 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

10 December 2020

Exam Rules

- Write all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of one 4 1/2" x 6" piece of paper filled with notes, programs, etc.
- While taking the exam, you may have with you pens and pencils, and your watch/clock.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College respects acts of academic dishonesty (i.e., plagiarism, cheating on examinations, obtaining unfair advantage, and fabrication of records and official documents) as serious offenses against the values of educational integrity. The College is committed to upholding the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

Enter your name and email address. Your name and email address will be printed on the front of the exam and will be used for identification.
Name:
Email:
Phone:
Signature:

Exam Options

Exam Times:

- **Default: Regular Time: Monday, 14 December, 9-11am.**
- **Alternate Time: Reading Day, Friday, 11 December, 8am-10am.**
- **Accessibility Testing: If you have not done so already, email me no later than 7 December.**
- **Survey for your choice will be available next lecture. No survey answer implies you will take the exam on 14 December.**

FINAL EXAM, VERSION 3
CS6112: Introduction to Computer Science
Hunter College, City University of New York

10 December 2020

Exam Rules

- Have all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of one 4 1/2" x 6" piece of paper filled with notes, programs, etc.
- While taking the exam, you may have with you pens and pencils, and your watch/clock.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College respects each student's ability to do as well as possible, including accommodations, alternative testing arrangements, and distribution of exams and official transcripts to various off-campus locations. The College is committed to providing the best possible learning environment for all students. Hunter College will provide any accommodations necessary to the Hunter College Academic Integrity Process.

• A handwritten name and number of academic dishonesty will be required on the form at the bottom of this exam.
Name:
Height:
Weight:
Signature:

Exam Options

Exam Times:

- **Default: Regular Time: Monday, 14 December, 9-11am.**
- **Alternate Time: Reading Day, Friday, 11 December, 8am-10am.**
- **Accessibility Testing: If you have not done so already, email me no later than 7 December.**
- **Survey for your choice will be available next lecture. No survey answer implies you will take the exam on 14 December.**

Grading Options:

FINAL EXAM, VERSION 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

10 December 2020

Exam Rules

- There are **two** exams. Your grade will be based on the **best** exam.
- The exam is closed book and closed notes with the exception of one 4 1/2" x 6" piece of paper filled with notes, programs, etc.
- While taking the exam, you may have with you pens and pencils, and your own clock.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College respects acts of academic dishonesty (i.e., plagiarism, cheating on examinations, obtaining unfair advantage, and fabrication of records and official documents) as serious offenses against the integrity of educational institutions. The College is committed to upholding the CUNY Policy on Academic Integrity and will pursue acts of academic dishonesty according to the Hunter College Academic Integrity Procedures.

<small>A comprehensive list of acts of academic dishonesty will be provided to the class at the beginning of the semester.</small>
Name: _____
Section: _____
Block: _____
Signature: _____

Exam Options

Exam Times:

- **Default: Regular Time: Monday, 14 December, 9-11am.**
- **Alternate Time: Reading Day, Friday, 11 December, 8am-10am.**
- **Accessibility Testing: If you have not done so already, email me no later than 7 December.**
- **Survey for your choice will be available next lecture. No survey answer implies you will take the exam on 14 December.**

FINAL EXAM, VERSION 3
CS6112: Introduction to Computer Science
Hunter College, City University of New York

10 December 2020

Exam Rules

- There are 100 questions. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of one 4 1/2" x 6" piece of paper filled with notes, programs, etc.
- While taking the exam, you may have with you pens and pencils, and your own clock.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

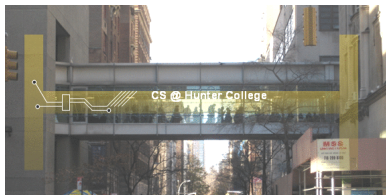
Hunter College respects acts of academic dishonesty (e.g., plagiarism, cheating on examinations, obtaining unfair advantages, and distribution of exams and official documents to persons off-site) through the Office of Academic Integrity. The College is committed to upholding the CUNY Policy on Academic Integrity and will pursue acts of academic dishonesty according to the Hunter College Academic Integrity Procedures.

Consent to the use of your work for academic purposes will be required to take the exam at Hunter College and will be used for educational purposes.
Name:
Height:
Weight:
Signature:

Grading Options:

- **Default: Letter Grade.**
- **Credit/NoCredit grade— check with academic advisor and fill out form by **25 November****

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab

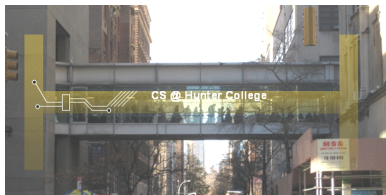
Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Optional - attend [live Lab Review on Wednesday 11-12:30pm](#)

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Optional - attend [live Lab Review on Wednesday 11-12:30pm](#)
- Take the Lab Quiz on Gradescope by 6pm on Wednesday

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Optional - attend [live Lab Review on Wednesday 11-12:30pm](#)
- Take the Lab Quiz on Gradescope by 6pm on Wednesday
- Submit this week's 3 programming assignments (programs 50-52)

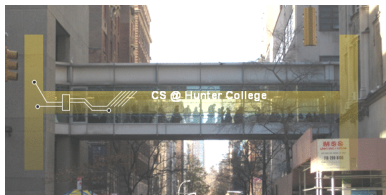
Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Optional - attend [live Lab Review on Wednesday 11-12:30pm](#)
- Take the Lab Quiz on Gradescope by 6pm on Wednesday
- Submit this week's 3 programming assignments (programs 50-52)
- At any point, visit our [Drop-In Tutoring 11am-5pm](#) for help!!!

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Optional - attend [live Lab Review on Wednesday 11-12:30pm](#)
- Take the Lab Quiz on Gradescope by 6pm on Wednesday
- Submit this week's 3 programming assignments (programs 50-52)
- At any point, visit our [Drop-In Tutoring 11am-5pm](#) for help!!!
- Take the Lecture Preview on Blackboard on Monday (or no later than 10am on Tuesday)