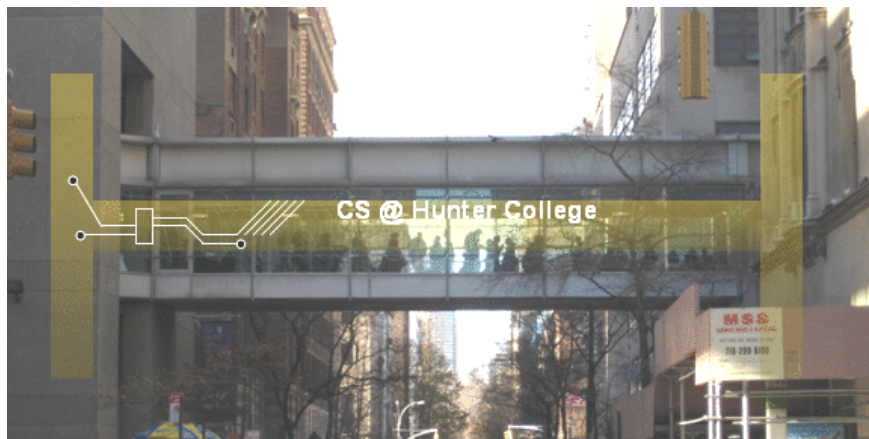


CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Frequently Asked Questions

From email and tutoring.

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

Frequently Asked Questions

From email and tutoring.

- **When is the final? Is there a review sheet?**
The official final is Monday, May 22 from 9-11 am.

Frequently Asked Questions

From email and tutoring.

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

The official final is Monday, May 22 from 9-11 am.

The early final exam (alternative date) is on Wednesday, May 17 in 1001G (time TBD).

Frequently Asked Questions

From email and tutoring.

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

The official final is Monday, May 22 from 9-11 am.

The early final exam (alternative date) is on Wednesday, May 17 in 1001G (time TBD).

Instead of a review sheet, we have:

Frequently Asked Questions

From email and tutoring.

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

The official final is Monday, May 22 from 9-11 am.

The early final exam (alternative date) is on Wednesday, May 17 in 1001G (time TBD).

Instead of a review sheet, we have:

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

Frequently Asked Questions

From email and tutoring.

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

The official final is Monday, May 22 from 9-11 am.

The early final exam (alternative date) is on Wednesday, May 17 in 1001G (time TBD).

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.

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

The official final is Monday, May 22 from 9-11 am.

The early final exam (alternative date) is on Wednesday, May 17 in 1001G (time TBD).

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.*
- ▶ *To help practice, there will be a mock exam during our last meeting on May 16.*

Frequently Asked Questions

From email and tutoring.

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

The official final is Monday, May 22 from 9-11 am.

The early final exam (alternative date) is on Wednesday, May 17 in 1001G (time TBD).

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.*
- ▶ *To help practice, there will be a mock exam during our last meeting on May 16.*
- ▶ *The mock exam will be run exactly like the real final.*

Frequently Asked Questions

From email and tutoring.

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

The official final is Monday, May 22 from 9-11 am.

The early final exam (alternative date) is on Wednesday, May 17 in 1001G (time TBD).

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.*
- ▶ *To help practice, there will be a mock exam during our last meeting on May 16.*
- ▶ *The mock exam will be run exactly like the real final.*
- ▶ *If you are already acquainted with the logistics you will have less stress during the real event.*

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.
- Stop 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

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)

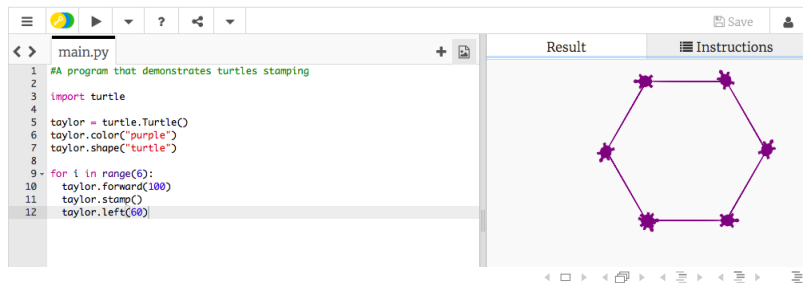
```
#This program prints: Hello, World!
```

← (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 editor displays the following Python code:

```
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 shows a purple hexagon with a turtle shape at each vertex, demonstrating the output of the code.

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.

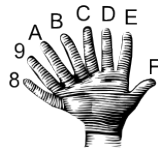
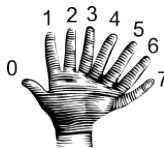
Week 2: variables, data types, more on loops & range()

- 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)
```

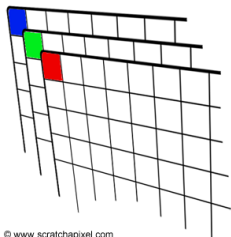
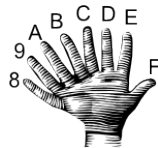
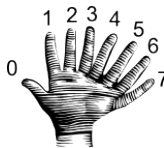

Week 3: colors, hex, slices, numpy & images

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








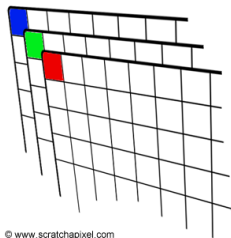
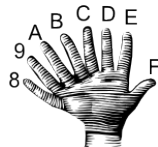
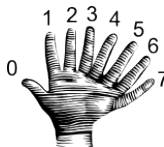
Week 3: colors, hex, slices, numpy & images

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.")

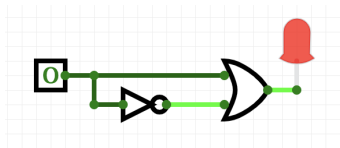
visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
    (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

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  
1830,202589,20535,9049,3023,7082,242278  
1840,312710,47613,34480,5344,10965,391114  
1850,515547,138882,18593,8032,15061,696115  
1860,813649,279122,32903,23593,25492,1174779  
1870,942282,419901,45648,37393,33029,1478183  
1880,1164673,599495,56559,51980,38991,1911698  
1890,1441216,838547,87050,88908,51693,2507414  
1900,1650093,1166582,152899,206507,67021,3437202  
1910,2331542,1634351,284041,430989,85969,4766883  
1920,2284183,2018256,448942,732018,116531,3620348  
1930,1867312,2560451,1079129,1265298,159346,6305446  
1940,1889924,2698285,1297634,1394711,174441,7454395  
1950,1960101,2738275,1550849,1452277,191555,7893257  
1960,1698281,2627319,1809578,1424815,221991,7781984  
1970,1539233,2602012,1996473,1471701,295443,7894862  
1980,1428285,2230936,1891325,1168972,352121,7077439  
1990,1487536,2300644,1951598,1203789,378977,7322564  
2000,1537195,2465326,2229379,1332450,443728,8008278  
2010,1648473,2504760,2230722,1385108,448736,8175133  
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,,,727,7481
1773,21883,3623,,,2847,28423
1790,33131,4548,6159,1181,3827,49447
1800,40515,5740,6642,1755,4543,79215
1810,46373,8303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,8049,3023,7082,242278
1840,312710,47613,34480,5344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32903,23593,25492,1174779
1870,942282,419801,45468,37393,33029,1478183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,115899,200507,67021,3437202
1910,2331542,1634351,284041,430989,85969,4768883
1920,2884183,2018356,448942,732018,116531,5620348
1930,3867312,2560451,1079129,1265298,159346,6950446
1940,4889924,2698285,1297634,1394711,174441,7454995
1950,5960101,2738275,1550849,1452277,191555,7893257
1960,6498281,2627319,1809578,1424815,221991,7781984
1970,5339233,2602012,1986473,1471701,295443,7094862
1980,4428285,2230936,1891325,1168972,352121,7077439
1990,3487536,2300644,1951598,1203789,378977,7322564
2000,3537195,2465326,2229379,1332450,443728,8008278
2010,3484873,2504760,2230722,1385108,448730,8175133
2015,3644518,2636738,2339150,1455444,474558,8550405
```

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,,727,7681
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
1830,202589,20535,8049,3023,7082,242278
1840,312710,47613,24480,3344,10965,399114
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
1900,1650093,1166581,152899,20507,67021,3437202
1910,2331542,1634351,284041,430989,85969,4766883
1920,2284183,2018356,448942,732018,116531,5620348
1930,3867312,2560451,1079129,1265298,159346,6950446
1940,4889924,2698285,1297634,1394711,174441,7454395
1950,1960101,2738275,1550849,1452177,191555,7893197
1960,4698281,2627319,1809578,1424815,221991,7781984
1970,5339233,2602012,1986473,1471701,295443,7894862
1980,4428285,2230936,1891325,1168972,352121,7077439
1990,4487536,2300644,1951598,1203789,378977,7322564
2000,4537195,2465326,2229379,1332450,443728,8008278
2010,4548473,2568760,2230728,1385108,448738,8175133
2015,4644518,2636738,2339150,1455444,474558,8550405
```

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
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
1840,312710,47613,10480,5344,10965,391114
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
1920,2284183,2018356,448942,732016,116531,5420048
1930,1867312,2560451,1079129,1565398,159346,6950446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738275,1550849,1452277,191555,7893257
1960,1698281,2627319,1809578,1424815,221991,7781986
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
2015,1644518,2636738,2339150,1455444,474558,8550405
```

nycHistPop.csv

In Lab 6

Week 6: structured data, pandas, & more design

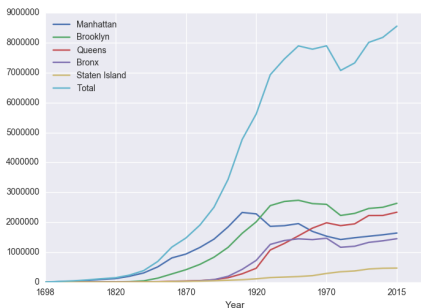
```
import matplotlib.pyplot as plt
import pandas as pd
```

```
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,,727,7481
1773,21883,3623,,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
1930,1867312,2560451,1079129,1565298,159346,6505446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738275,1500849,1452277,191559,7893257
1960,1698281,2627319,1809578,1424815,221991,7781984
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
```

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



nycHistPop.csv

In Lab 6

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

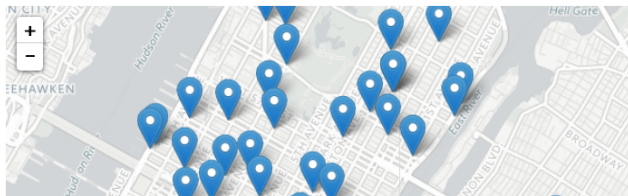
```
def totalWithTax(food, tip):  
    total = 0  
    tax = 0.0875  
    total = food + food * tax  
    total = total + tip  
    return(total)  
  
lunch = float(input('Enter lunch total: '))  
lTip = float(input('Enter lunch tip: '))  
lTotal = totalWithTax(lunch, lTip)  
print('Lunch total is',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



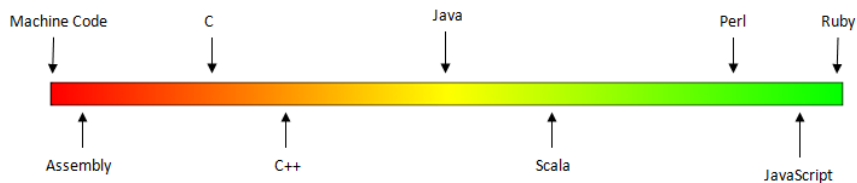
- Input/Output (I/O): `input()` and `print()`;
pandas for CSV files
- Types:
 - ▶ Primitive: `int`, `float`, `bool`, `string`;
 - ▶ Container: lists (but not dictionaries/hashtes or tuples)
- Objects: turtles (used but did not design our own)
- Loops: definite & indefinite
- Conditionals: `if-elif-else`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

Today's Topics



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

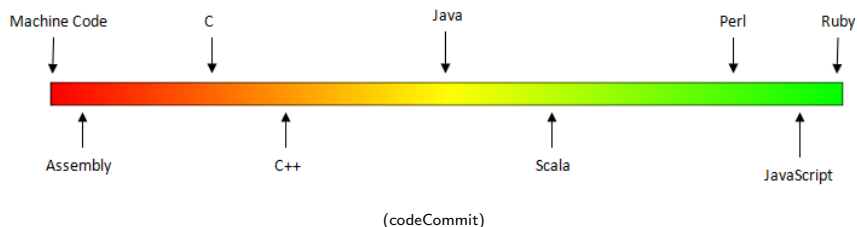
Low-Level vs. High-Level Languages



(codeCommit)

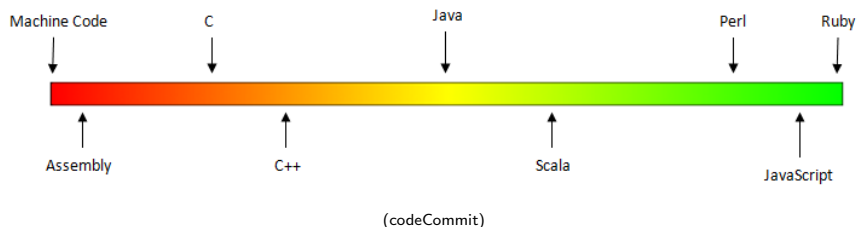
- Can view programming languages on a continuum.

Low-Level vs. High-Level Languages



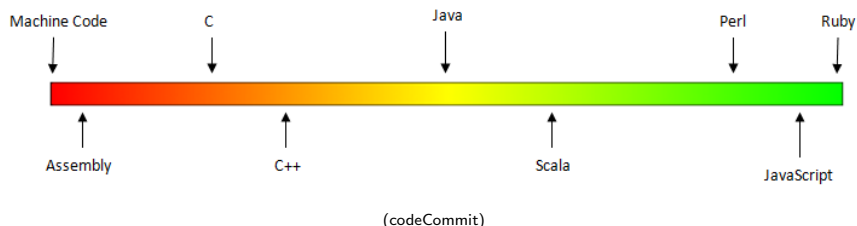
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

Low-Level vs. High-Level Languages



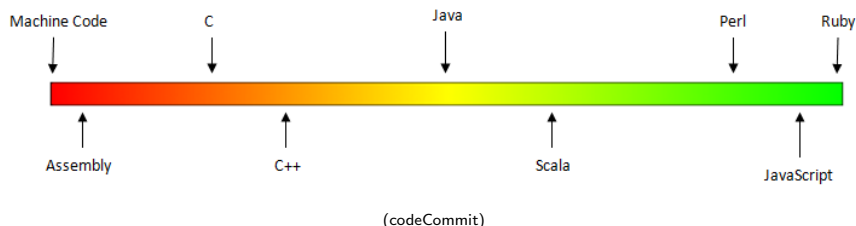
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

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

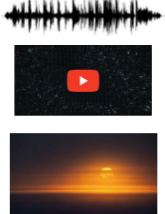


- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
- Those that have strong abstraction (allow programming paradigms independent of the machine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



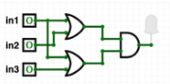
Dies ist ein Blindtest. An ihm lässt sich vieles über die Schrift ablesen, in der er gesetzt ist. Auf den ersten Blick wird der Grauwert der Schriftfläche sichtbar. Dann kann man prüfen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt. Dies ist ein Blindtest. An ihm lässt sich



Data & Instructions

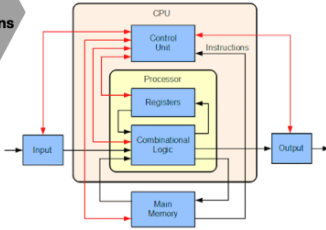


Data & Instructions

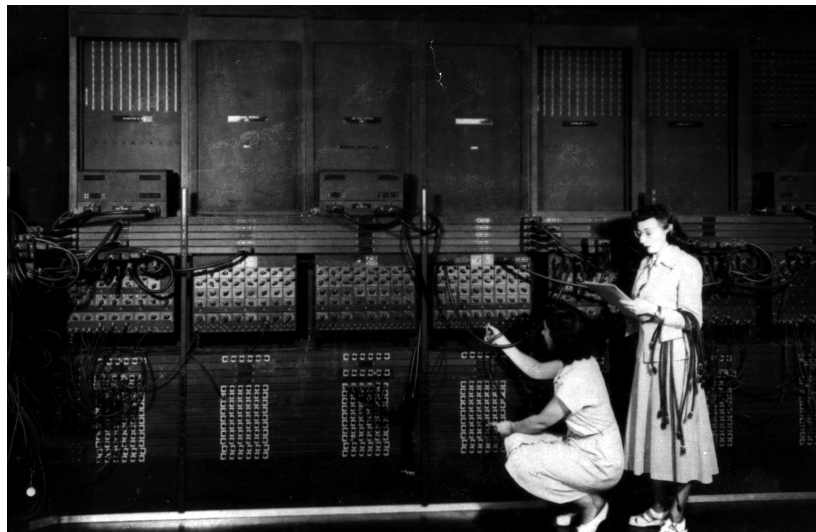


Circuits (switches)
On/Off 1/0 Logic
Billions of switches/bits

```
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
R 002002 1B      CLC
R 002003 FB      SED
R 002004 09 34 12  LDR #01234
R 002007 09 21 43  RLC #04321
R 002009 0F 03 7F 01  STA #017F03
R 00200C 00      CLD
R 00200F 02 30      SEP #K30
R 002011 0B      BRK
R 2012

P  PC  Mem32C  A  X  Y  SP  BP  BB
: 00 0012  00110000 0000 0000 0002 C7FF 0000 00
$ 200B
BREAK
P  PC  Mem32C  A  X  Y  SP  BP  BB
: 00 2013  00110000 5555 0000 0002 C7FF 0000 00
n 1103 7F03
00FF03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(wiki)

Machine Language

```
002000 C2 30 REP #430
002002 1B CLC
002003 F8 SED
002004 09 34 12 LBR #01234
002007 09 21 43 RBC #04321
00200A 0F 03 7F 01 STA #017F03
00200C 00 CLD
00200F E2 30 SEP #430
002011 00 BRK
002012

PB PC Mem32C A X Y SP BP
: 00 2012 00110000 0000 0000 0002 C7FF 0000 00
$ 2000

BREAK

PB PC Mem32C A X Y SP BP
: 00 2013 00110000 5555 0000 0002 C7FF 0000 00
n 1103 7F03
007F03 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 1B CLC
002003 F8 SED
002004 00 34 12 LSH #01234
002007 00 21 43 RLC #04321
002009 0F 03 7F 01 STA #017F03
00200C 00 CLD
00200F 02 30 SEP #430
002011 00 BRK
02012

PB PC Mem32C A X Y SP BP
: 00 2012 00110000 0000 0000 0000 0000 0000 0000 0000
$ 2000
BREAK

PB PC Mem32C A X Y SP BP
: 00 2013 00110000 5555 0000 0000 0000 0000 0000 0000
n 1103 7F03
007F03 55 00 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.

Machine Language

```
002000 c2 30      REP #430
002002 18          CLC
002003 f8          SED
002004 49 34 12    LBR #01234
002007 69 21 43    RBC #04321
002009 6f 03 7f 01  STA #017f03
00200c 00          CLD
00200f e2 30      SEP #430
002011 00          BRK
02012

P  PC Mem32C  A  X  Y  SP  BP  BB
: 00 0012  00110000 0000 0000 0000 0000 0000 00
$ 2000
BREAK
P  PC Mem32C  A  X  Y  SP  BP  BB
: 00 2013  00110000 5555 0000 0000 0000 0000 00
n 1103 7f03
00f703 55 55 00 00 00 00 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 | dis | ShowHide Demos

Additional Doubler | Stop | Looper | Stack Test | Hello World

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

Debug

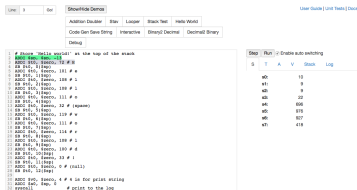
```
1 # Store 'hello world!' at the top of the stack
2 ADDI $a0, $zero, 32 # $0
3 SD $a0, 0($sp)
4 ADDI $d0, $zero, 191 # e
5 SD $d0, 4($sp)
6 SD $a0, 8($sp)
7 ADDI $d0, $zero, 198 # l
8 SD $d0, 12($sp)
9 ADDI $d0, $zero, 199 # l
10 SD $d0, 16($sp)
11 ADDI $d0, $zero, 111 # o
12 SD $d0, 20($sp)
13 ADDI $d0, $zero, 92 # (space)
14 SD $d0, 24($sp)
15 ADDI $d0, $zero, 113 # w
16 SD $d0, 28($sp)
17 ADDI $d0, $zero, 114 # a
18 SD $d0, 32($sp)
19 ADDI $d0, $zero, 114 # a
20 SD $d0, 36($sp)
21 ADDI $d0, $zero, 108 # l
22 SD $d0, 40($sp)
23 ADDI $d0, $zero, 103 # d
24 SD $d0, 44($sp)
25 ADDI $d0, $zero, 33 # !
26 SD $d0, 48($sp)
27 ADDI $d0, $zero, 0 # (null)
28 SD $d0, 52($sp)
29
30 ADDI $v0, $zero, 6 # 6 in for print string
31 ADDI $a0, $v0, 0 # print to the log
32 syscall
```

Step | Run | Enable auto switching

S	T	A	V	Stack	Log
				a0:	10
				a1:	9
				a2:	9
				a3:	22
				a4:	995
				a5:	976
				a6:	927
				a7:	418

(Demo with WeMIPS)

MIPS Commands



The screenshot shows the MIPS simulator interface. On the left, there is a text area containing assembly code. On the right, there is a window titled "Registers" showing the current state of the registers.

```
# Show "Hello world" as the top of the stack
1  ADDI $PC, $ZERO, 12 # R
2  $PC: 12000
3  ADDI $A0, $ZERO, 10 # R
4  $A0: 10000
5  $PC: 12004
6  SB $A0, 0($PC)
7  ADDI $PC, $PC, 100 # I
8  $PC: 12100
9  ADDI $A0, $ZERO, 11 # R
10 $A0: 11000
11 $PC: 12104
12 ADDI $A0, $ZERO, 12 # (memory)
13 $A0: 12000
14 $PC: 12108
15 SB $A0, 4($PC)
16 ADDI $A0, $ZERO, 11 # R
17 $A0: 11000
18 ADDI $A0, $ZERO, 114 # R
19 $A0: 114000
20 $PC: 12112
21 ADDI $A0, $ZERO, 100 # I
22 $A0: 100000
23 ADDI $A0, $ZERO, 100 # R
24 $A0: 100000
25 SB $A0, 0($PC)
26 $PC: 12116
27 ADDI $A0, $ZERO, 10 # I
28 $A0: 100000
29 $PC: 12120
30 ADDI $A0, $ZERO, 0 # (null)
31 $A0: 000000
32 $PC: 12124
33 ADDI $PC, $ZERO, 4 # R in the print ending
34 ADDI $PC, $PC, 4
35 syscall
# print to the log
```

The Register window shows the following state:

Register	Value
\$0	10
\$1	0
\$2	0
\$3	100
\$4	114000
\$5	100000
\$6	100000
\$7	0

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

MIPS Commands

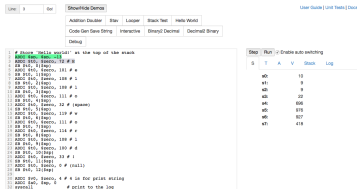
The screenshot shows a MIPS simulator interface. On the left, there is a text area containing assembly code. On the right, there is a register window titled 'Stop Run' with a sub-option 'Create auto-switching'. The register window displays the values of registers \$0 through \$31.

```
# Show "Hello world" as the top of the stack
ADDI $0, $zero, 12 # R
SB $0, 0($0)
ADDI $0, $zero, 181 # R
SB $0, 1($0)
ADDI $0, $zero, 100 # L
SB $0, 2($0)
ADDI $0, $zero, 111 # R
SB $0, 3($0)
ADDI $0, $zero, 12 # (same)
SB $0, 4($0)
ADDI $0, $zero, 110 # R
SB $0, 5($0)
ADDI $0, $zero, 111 # R
SB $0, 6($0)
ADDI $0, $zero, 114 # R
SB $0, 7($0)
ADDI $0, $zero, 100 # L
SB $0, 8($0)
ADDI $0, $zero, 100 # R
SB $0, 9($0)
ADDI $0, $zero, 10 # L
SB $0, 10($0)
ADDI $0, $zero, 0 # (null)
SB $0, 11($0)
ADDI $0, $zero, 4 # 4 in the print string
ADDI $0, $zero, 8
syscall
```

\$	T	A	V	Stack	Log
\$0				10	
\$1				8	
\$2				8	
\$3				22	
\$4				100	
\$5				110	
\$6				111	
\$7				114	

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

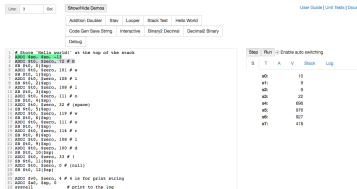
MIPS Commands



The screenshot shows a MIPS simulator interface. On the left, there is a text area containing assembly code. The code starts with a comment: `# Show "hello world" at the top of the stack`. It then uses `ADDI $t0, $zero, 12` to load the value 12 into register `$t0`. This is followed by a loop of instructions: `SB $t0, 0($t0)`, `ADDI $t0, $zero, 100 # n`, `SB $t0, 0($t0)`, `ADDI $t0, $zero, 100 # i`, `SB $t0, 0($t0)`, `ADDI $t0, $zero, 111 # n`, `SB $t0, 0($t0)`, `ADDI $t0, $zero, 12 # (n+1)`, `SB $t0, 0($t0)`, `ADDI $t0, $zero, 120 # n`, `SB $t0, 0($t0)`, `ADDI $t0, $zero, 111 # n`, `SB $t0, 0($t0)`, `ADDI $t0, $zero, 114 # n`, `SB $t0, 0($t0)`, `ADDI $t0, $zero, 100 # i`, `SB $t0, 0($t0)`, `ADDI $t0, $zero, 100 # n`, `SB $t0, 0($t0)`, `ADDI $t0, $zero, 10 # i`, `SB $t0, 0($t0)`, `ADDI $t0, $zero, 0 # (n+1)`, `SB $t0, 0($t0)`, `ADDI $t0, $zero, 0 # (n+1)`, `SB $t0, 0($t0)`, `ADDI $t0, $zero, 4 # n for print ending`, `ADDI $t0, $zero, 0`, and finally `syscall`. On the right, there is a register window titled "Registers" with columns for "S", "T", "A", "V", "Stack", and "Log". The registers shown are `$0` (0), `$1` (0), `$2` (0), `$3` (0), `$4` (0), `$5` (0), `$6` (0), `$7` (0), and `$8` (0).

- **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

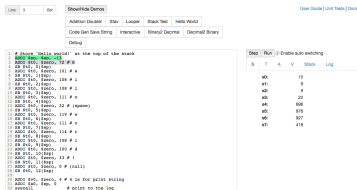


The screenshot shows a MIPS simulator interface. On the left, there is a text area containing assembly code. On the right, there is a register window titled 'Stop Run' with a sub-option 'Create auto-switching'. The register window displays the following values:

S	T	A	V	Stack	Log
0	0	0	0	0	
1	0	0	0	0	
2	0	0	0	0	
3	0	0	0	0	
4	0	0	0	0	
5	0	0	0	0	
6	0	0	0	0	
7	0	0	0	0	
8	0	0	0	0	
9	0	0	0	0	
10	0	0	0	0	
11	0	0	0	0	
12	0	0	0	0	
13	0	0	0	0	
14	0	0	0	0	
15	0	0	0	0	
16	0	0	0	0	
17	0	0	0	0	
18	0	0	0	0	
19	0	0	0	0	
20	0	0	0	0	
21	0	0	0	0	
22	0	0	0	0	
23	0	0	0	0	
24	0	0	0	0	
25	0	0	0	0	
26	0	0	0	0	
27	0	0	0	0	
28	0	0	0	0	
29	0	0	0	0	
30	0	0	0	0	
31	0	0	0	0	

- **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

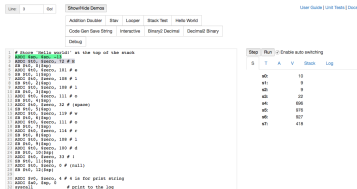


The screenshot shows a MIPS simulator interface. On the left, there is a text area containing assembly code. The code includes comments and instructions for adding values to registers \$s0 through \$s15. On the right, there is a register window titled 'Registers' with columns for 'S', 'T', 'A', 'V', 'Stack', and 'Log'. The register values are as follows:

Register	Value
\$0	10
\$1	9
\$2	8
\$3	7
\$4	6
\$5	5
\$6	4
\$7	3
\$8	2
\$9	1
\$10	0
\$11	0
\$12	0
\$13	0
\$14	0
\$15	0

- **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 with a text editor on the left containing assembly code and a register window on the right. The register window displays the values of registers \$0 through \$31.

```
# Show "Hello world" at the top of the stack
addi $0, $zero, 12 # R
sb $0, 0($zero)
addi $0, $zero, 10 # R
sb $0, 1($zero)
addi $0, $zero, 100 # I
sb $0, 2($zero)
addi $0, $zero, 111 # R
sb $0, 3($zero)
addi $0, $zero, 12 # (same)
sb $0, 4($zero)
addi $0, $zero, 111 # R
sb $0, 5($zero)
addi $0, $zero, 114 # R
sb $0, 6($zero)
addi $0, $zero, 100 # I
sb $0, 7($zero)
addi $0, $zero, 100 # R
sb $0, 8($zero)
addi $0, $zero, 10 # I
sb $0, 9($zero)
addi $0, $zero, 0 # (null)
sb $0, 10($zero)
addi $0, $zero, 4 # 4 in the print string
addi $0, $zero, 0
syscall
```

\$	T	A	V	Stack	Log
\$0				10	
\$1				8	
\$2				8	
\$3				22	
\$4				100	
\$5				111	
\$6				114	
\$7				100	
\$8				100	
\$9				10	
\$10				0	
\$11				4	
\$12				0	
\$13				0	
\$14				0	
\$15				0	
\$16				0	
\$17				0	
\$18				0	
\$19				0	
\$20				0	
\$21				0	
\$22				0	
\$23				0	
\$24				0	
\$25				0	
\$26				0	
\$27				0	
\$28				0	
\$29				0	
\$30				0	
\$31				0	

- **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 the MIPS simulator interface. On the left, there is a text area containing assembly code. On the right, there is a register window showing the values of registers \$0 through \$31.

```
# Show "Hello world" at the top of the screen
addi $v0, $zero, 12 # R
sb $v0, $zero
addi $v0, $zero, 10 # R
sb $v0, $zero
addi $v0, $zero, 100 # I
sb $v0, $zero
addi $v0, $zero, 111 # R
sb $v0, $zero
addi $v0, $zero, 12 # (imm)
sb $v0, $zero
addi $v0, $zero, 111 # R
sb $v0, $zero
addi $v0, $zero, 114 # R
sb $v0, $zero
addi $v0, $zero, 100 # I
sb $v0, $zero
addi $v0, $zero, 10 # I
sb $v0, $zero
addi $v0, $zero, 0 # (null)
sb $v0, $zero
addi $v0, $zero, 4 # 4 for print ending
addi $v0, $zero, 0
syscall
# print to the top
```

\$	T	A	V	Stack	Log
\$0			10		
\$1			0		
\$2			0		
\$3			22		
\$4			100		
\$5			111		
\$6			12		
\$7			111		
\$8			114		
\$9			100		
\$10			10		
\$11			0		
\$12			4		
\$13			0		
\$14			0		
\$15			0		
\$16			0		
\$17			0		
\$18			0		
\$19			0		
\$20			0		
\$21			0		
\$22			0		
\$23			0		
\$24			0		
\$25			0		
\$26			0		
\$27			0		
\$28			0		
\$29			0		
\$30			0		
\$31			0		

- **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 displays the MIPS simulator interface. The main window shows assembly code with comments: `# Show 'hello world' at the top of the stack`. The code includes instructions for adding immediate values to registers `$s0` through `$s4`, and for printing the contents of `$s0`. The register window on the right shows the values for registers `$0` through `$31`.

```
# Show 'hello world' at the top of the stack
addi $s0, $zero, 12 # R
addi $s1, $zero, 181 # R
addi $s2, $zero, 100 # R
addi $s3, $zero, 100 # R
addi $s4, $zero, 100 # R
addi $s0, $zero, 100 # R
addi $s0, $zero, 111 # R
addi $s0, $zero, 12 # (again)
addi $s0, $zero, 100 # R
addi $s0, $zero, 111 # R
addi $s0, $zero, 114 # R
addi $s0, $zero, 100 # R
addi $s0, $zero, 100 # R
addi $s0, $zero, 10 # R
addi $s0, $zero, 0 # (null)
addi $s0, $zero, 4 # in the print window
addi $s0, $zero, 4
syscall
```

\$	T	A	V	Stack	Log
\$0				10	
\$1				9	
\$2				8	
\$3				7	
\$4				6	
\$5				5	
\$6				4	
\$7				3	
\$8				2	
\$9				1	
\$10				0	
\$11				-1	
\$12				-2	
\$13				-3	
\$14				-4	
\$15				-5	
\$16				-6	
\$17				-7	
\$18				-8	
\$19				-9	
\$20				-10	
\$21				-11	
\$22				-12	
\$23				-13	
\$24				-14	
\$25				-15	
\$26				-16	
\$27				-17	
\$28				-18	
\$29				-19	
\$30				-20	
\$31				-21	

- **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. On the left, there is a list of assembly instructions:


```

1 # Show 'hello world' at the top of the stack
2 ADDI $s0, $zero, 12 # R
3 SB $s0, 0($zero)
4 ADDI $s1, $zero, 100 # R
5 SB $s1, 1($zero)
6 ADDI $s2, $zero, 100 # I
7 SB $s2, 2($zero)
8 ADDI $s3, $zero, 111 # R
9 SB $s3, 3($zero)
10 ADDI $s4, $zero, 12 # (imm)
11 SB $s4, 4($zero)
12 ADDI $s5, $zero, 111 # R
13 SB $s5, 5($zero)
14 ADDI $s6, $zero, 114 # R
15 SB $s6, 6($zero)
16 ADDI $s7, $zero, 100 # I
17 SB $s7, 7($zero)
18 ADDI $s8, $zero, 100 # I
19 SB $s8, 8($zero)
20 ADDI $s9, $zero, 10 # I
21 SB $s9, 9($zero)
22 ADDI $s10, $zero, 0 # (null)
23 SB $s10, 10($zero)
24 ADDI $s11, $zero, 4 # 4 in the print string
25 ADDI $s12, $s0, 4
26 syscall
    
```

 On the right, a register window is visible with columns for \$, T, A, V, Stack, and Log. The \$ register column shows values: \$0: 10, \$1: 0, \$2: 0, \$3: 100, \$4: 12, \$5: 111, \$6: 114, \$7: 100, \$8: 100, \$9: 10, \$10: 0, \$11: 4, \$12: 4.

- **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

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 $a0, 0($sp)
4 ADDI $d0, $zero, 191 # e
5 SD $d0, 4($sp)
6 ADDI $d0, $zero, 108 # l
7 SD $d0, 8($sp)
8 ADDI $d0, $zero, 108 # l
9 SD $d0, 12($sp)
10 ADDI $d0, $zero, 111 # o
11 SD $d0, 16($sp)
12 ADDI $d0, $zero, 32 # (space)
13 SD $d0, 20($sp)
14 ADDI $d0, $zero, 113 # w
15 SD $d0, 24($sp)
16 ADDI $d0, $zero, 113 # w
17 SD $d0, 28($sp)
18 SD $a0, 32($sp)
19 ADDI $d0, $zero, 114 # z
20 SD $d0, 36($sp)
21 ADDI $d0, $zero, 108 # l
22 SD $d0, 40($sp)
23 ADDI $d0, $zero, 108 # l
24 SD $d0, 44($sp)
25 ADDI $d0, $zero, 33 # i
26 SD $d0, 48($sp)
27 ADDI $d0, $zero, 0 # (null)
28 SD $d0, 52($sp)
29 ADDI $v0, $zero, 6 # 4 in for print string
30 ADDI $a0, $a0, 0 # print to the log
31 syscall
```

Step	Run	Enable auto switching			
S	T	A	V	Stack	Log
a0				10	
a1				9	
a2				9	
a3				22	
a4				905	
a5				976	
a6				927	
a7				418	

(Demo with WeMIPS)

Today's Topics



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

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 image shows a debugger window with two panes. The left pane displays assembly instructions, including a loop structure with instructions like `mov rax, 10`, `dec rax`, `cmp rax, 0`, and `je loop_start`. The right pane shows the state of registers, with `rax` containing `00000000` and `rbp` containing `00000000`.

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.



A screenshot of an IDE window titled 'AssemblyView'. The main pane displays assembly code with several lines highlighted in green. The code includes labels like 'start', 'loop_start', and 'loop_end', and instructions such as 'mov', 'add', 'sub', 'cmp', 'jmp', and 'ret'. A secondary pane on the right shows a disassembly view with a list of instructions and their corresponding memory addresses.

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.



```
00000000: 74 05 00 00 <JMP 5>
00000004: 31 05 00 00 <MOV EAX, 5>
00000008: 90 00 00 00 <NOP>
0000000c: 31 06 00 00 <MOV EAX, 6>
00000010: 90 00 00 00 <NOP>
00000014: 31 07 00 00 <MOV EAX, 7>
00000018: 90 00 00 00 <NOP>
0000001c: 31 08 00 00 <MOV EAX, 8>
00000020: 90 00 00 00 <NOP>
00000024: 31 09 00 00 <MOV EAX, 9>
00000028: 90 00 00 00 <NOP>
0000002c: 31 0a 00 00 <MOV EAX, 10>
00000030: 90 00 00 00 <NOP>
00000034: 31 0b 00 00 <MOV EAX, 11>
00000038: 90 00 00 00 <NOP>
0000003c: 31 0c 00 00 <MOV EAX, 12>
00000040: 90 00 00 00 <NOP>
00000044: 31 0d 00 00 <MOV EAX, 13>
00000048: 90 00 00 00 <NOP>
0000004c: 31 0e 00 00 <MOV EAX, 14>
00000050: 90 00 00 00 <NOP>
00000054: 31 0f 00 00 <MOV EAX, 15>
00000058: 90 00 00 00 <NOP>
0000005c: 31 10 00 00 <MOV EAX, 16>
00000060: 90 00 00 00 <NOP>
00000064: 31 11 00 00 <MOV EAX, 17>
00000068: 90 00 00 00 <NOP>
0000006c: 31 12 00 00 <MOV EAX, 18>
00000070: 90 00 00 00 <NOP>
00000074: 31 13 00 00 <MOV EAX, 19>
00000078: 90 00 00 00 <NOP>
0000007c: 31 14 00 00 <MOV EAX, 20>
00000080: 90 00 00 00 <NOP>
00000084: 31 15 00 00 <MOV EAX, 21>
00000088: 90 00 00 00 <NOP>
0000008c: 31 16 00 00 <MOV EAX, 22>
00000090: 90 00 00 00 <NOP>
00000094: 31 17 00 00 <MOV EAX, 23>
00000098: 90 00 00 00 <NOP>
0000009c: 31 18 00 00 <MOV EAX, 24>
000000a0: 90 00 00 00 <NOP>
000000a4: 31 19 00 00 <MOV EAX, 25>
000000a8: 90 00 00 00 <NOP>
000000ac: 31 1a 00 00 <MOV EAX, 26>
000000b0: 90 00 00 00 <NOP>
000000b4: 31 1b 00 00 <MOV EAX, 27>
000000b8: 90 00 00 00 <NOP>
000000bc: 31 1c 00 00 <MOV EAX, 28>
000000c0: 90 00 00 00 <NOP>
000000c4: 31 1d 00 00 <MOV EAX, 29>
000000c8: 90 00 00 00 <NOP>
000000cc: 31 1e 00 00 <MOV EAX, 30>
000000d0: 90 00 00 00 <NOP>
000000d4: 31 1f 00 00 <MOV EAX, 31>
000000d8: 90 00 00 00 <NOP>
000000dc: 31 20 00 00 <MOV EAX, 32>
000000e0: 90 00 00 00 <NOP>
000000e4: 31 21 00 00 <MOV EAX, 33>
000000e8: 90 00 00 00 <NOP>
000000ec: 31 22 00 00 <MOV EAX, 34>
000000f0: 90 00 00 00 <NOP>
000000f4: 31 23 00 00 <MOV EAX, 35>
000000f8: 90 00 00 00 <NOP>
000000fc: 31 24 00 00 <MOV EAX, 36>
00000100: 90 00 00 00 <NOP>
00000104: 31 25 00 00 <MOV EAX, 37>
00000108: 90 00 00 00 <NOP>
0000010c: 31 26 00 00 <MOV EAX, 38>
00000110: 90 00 00 00 <NOP>
00000114: 31 27 00 00 <MOV EAX, 39>
00000118: 90 00 00 00 <NOP>
0000011c: 31 28 00 00 <MOV EAX, 40>
00000120: 90 00 00 00 <NOP>
00000124: 31 29 00 00 <MOV EAX, 41>
00000128: 90 00 00 00 <NOP>
0000012c: 31 2a 00 00 <MOV EAX, 42>
00000130: 90 00 00 00 <NOP>
00000134: 31 2b 00 00 <MOV EAX, 43>
00000138: 90 00 00 00 <NOP>
0000013c: 31 2c 00 00 <MOV EAX, 44>
00000140: 90 00 00 00 <NOP>
00000144: 31 2d 00 00 <MOV EAX, 45>
00000148: 90 00 00 00 <NOP>
0000014c: 31 2e 00 00 <MOV EAX, 46>
00000150: 90 00 00 00 <NOP>
00000154: 31 2f 00 00 <MOV EAX, 47>
00000158: 90 00 00 00 <NOP>
0000015c: 31 30 00 00 <MOV EAX, 48>
00000160: 90 00 00 00 <NOP>
00000164: 31 31 00 00 <MOV EAX, 49>
00000168: 90 00 00 00 <NOP>
0000016c: 31 32 00 00 <MOV EAX, 50>
00000170: 90 00 00 00 <NOP>
00000174: 31 33 00 00 <MOV EAX, 51>
00000178: 90 00 00 00 <NOP>
0000017c: 31 34 00 00 <MOV EAX, 52>
00000180: 90 00 00 00 <NOP>
00000184: 31 35 00 00 <MOV EAX, 53>
00000188: 90 00 00 00 <NOP>
0000018c: 31 36 00 00 <MOV EAX, 54>
00000190: 90 00 00 00 <NOP>
00000194: 31 37 00 00 <MOV EAX, 55>
00000198: 90 00 00 00 <NOP>
0000019c: 31 38 00 00 <MOV EAX, 56>
000001a0: 90 00 00 00 <NOP>
000001a4: 31 39 00 00 <MOV EAX, 57>
000001a8: 90 00 00 00 <NOP>
000001ac: 31 3a 00 00 <MOV EAX, 58>
000001b0: 90 00 00 00 <NOP>
000001b4: 31 3b 00 00 <MOV EAX, 59>
000001b8: 90 00 00 00 <NOP>
000001bc: 31 3c 00 00 <MOV EAX, 60>
000001c0: 90 00 00 00 <NOP>
000001c4: 31 3d 00 00 <MOV EAX, 61>
000001c8: 90 00 00 00 <NOP>
000001cc: 31 3e 00 00 <MOV EAX, 62>
000001d0: 90 00 00 00 <NOP>
000001d4: 31 3f 00 00 <MOV EAX, 63>
000001d8: 90 00 00 00 <NOP>
000001dc: 31 40 00 00 <MOV EAX, 64>
000001e0: 90 00 00 00 <NOP>
000001e4: 31 41 00 00 <MOV EAX, 65>
000001e8: 90 00 00 00 <NOP>
000001ec: 31 42 00 00 <MOV EAX, 66>
000001f0: 90 00 00 00 <NOP>
000001f4: 31 43 00 00 <MOV EAX, 67>
000001f8: 90 00 00 00 <NOP>
000001fc: 31 44 00 00 <MOV EAX, 68>
00000200: 90 00 00 00 <NOP>
00000204: 31 45 00 00 <MOV EAX, 69>
00000208: 90 00 00 00 <NOP>
0000020c: 31 46 00 00 <MOV EAX, 70>
00000210: 90 00 00 00 <NOP>
00000214: 31 47 00 00 <MOV EAX, 71>
00000218: 90 00 00 00 <NOP>
0000021c: 31 48 00 00 <MOV EAX, 72>
00000220: 90 00 00 00 <NOP>
00000224: 31 49 00 00 <MOV EAX, 73>
00000228: 90 00 00 00 <NOP>
0000022c: 31 4a 00 00 <MOV EAX, 74>
00000230: 90 00 00 00 <NOP>
00000234: 31 4b 00 00 <MOV EAX, 75>
00000238: 90 00 00 00 <NOP>
0000023c: 31 4c 00 00 <MOV EAX, 76>
00000240: 90 00 00 00 <NOP>
00000244: 31 4d 00 00 <MOV EAX, 77>
00000248: 90 00 00 00 <NOP>
0000024c: 31 4e 00 00 <MOV EAX, 78>
00000250: 90 00 00 00 <NOP>
00000254: 31 4f 00 00 <MOV EAX, 79>
00000258: 90 00 00 00 <NOP>
0000025c: 31 50 00 00 <MOV EAX, 80>
00000260: 90 00 00 00 <NOP>
00000264: 31 51 00 00 <MOV EAX, 81>
00000268: 90 00 00 00 <NOP>
0000026c: 31 52 00 00 <MOV EAX, 82>
00000270: 90 00 00 00 <NOP>
00000274: 31 53 00 00 <MOV EAX, 83>
00000278: 90 00 00 00 <NOP>
0000027c: 31 54 00 00 <MOV EAX, 84>
00000280: 90 00 00 00 <NOP>
00000284: 31 55 00 00 <MOV EAX, 85>
00000288: 90 00 00 00 <NOP>
0000028c: 31 56 00 00 <MOV EAX, 86>
00000290: 90 00 00 00 <NOP>
00000294: 31 57 00 00 <MOV EAX, 87>
00000298: 90 00 00 00 <NOP>
0000029c: 31 58 00 00 <MOV EAX, 88>
000002a0: 90 00 00 00 <NOP>
000002a4: 31 59 00 00 <MOV EAX, 89>
000002a8: 90 00 00 00 <NOP>
000002ac: 31 5a 00 00 <MOV EAX, 90>
000002b0: 90 00 00 00 <NOP>
000002b4: 31 5b 00 00 <MOV EAX, 91>
000002b8: 90 00 00 00 <NOP>
000002bc: 31 5c 00 00 <MOV EAX, 92>
000002c0: 90 00 00 00 <NOP>
000002c4: 31 5d 00 00 <MOV EAX, 93>
000002c8: 90 00 00 00 <NOP>
000002cc: 31 5e 00 00 <MOV EAX, 94>
000002d0: 90 00 00 00 <NOP>
000002d4: 31 5f 00 00 <MOV EAX, 95>
000002d8: 90 00 00 00 <NOP>
000002dc: 31 60 00 00 <MOV EAX, 96>
000002e0: 90 00 00 00 <NOP>
000002e4: 31 61 00 00 <MOV EAX, 97>
000002e8: 90 00 00 00 <NOP>
000002ec: 31 62 00 00 <MOV EAX, 98>
000002f0: 90 00 00 00 <NOP>
000002f4: 31 63 00 00 <MOV EAX, 99>
000002f8: 90 00 00 00 <NOP>
000002fc: 31 64 00 00 <MOV EAX, 100>
00000300: 90 00 00 00 <NOP>
```


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`.



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.



```

1:  .text
2:  .globl main
3:  .type main, @function
4:  main:
5:  .li $v0, 4
6:  .li $a0, "hello\n"
7:  syscall
8:  .li $v0, 10
9:  syscall
10: .end main

```

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 loop that prints the numbers 1 through 10.
# The loop is implemented using a branch instruction.
# The label 'loop_start' marks the beginning of the loop.
loop_start:
    li $t0, 1           # Load the value 1 into register $t0.
    print_int $t0      # Print the value in register $t0.
    addi $t0, $t0, 1   # Increment the value in register $t0 by 1.
    bne $t0, $zero, loop_start  # Branch to loop_start if $t0 is not equal to zero.

```

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

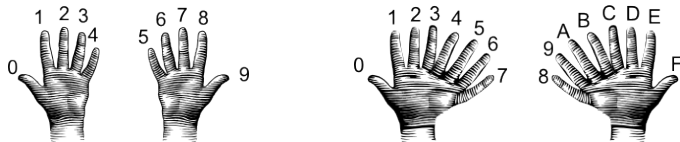
abcdefghijklmnopqrstuvwxyz

Today's Topics



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

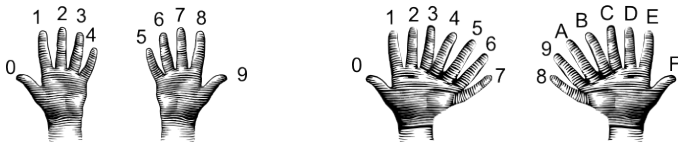
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

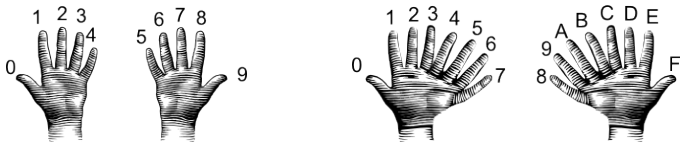
Hexadecimal to Decimal: Converting Between Bases



(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.

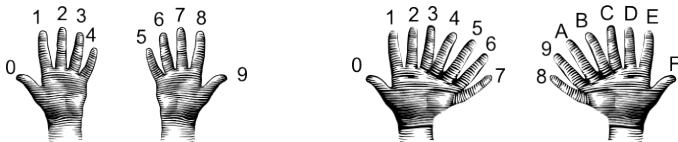
Hexadecimal to Decimal: Converting Between Bases



(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?

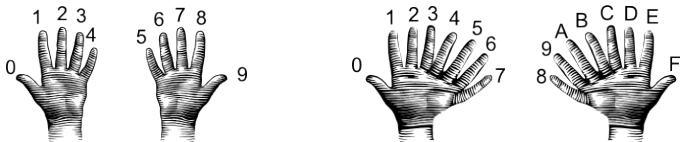
Hexadecimal to Decimal: Converting Between Bases



(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.

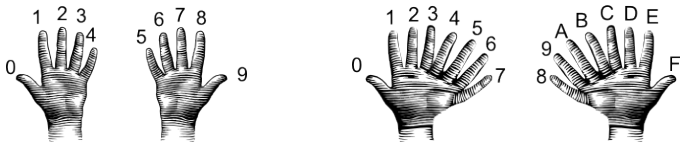
Hexadecimal to Decimal: Converting Between Bases



(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.

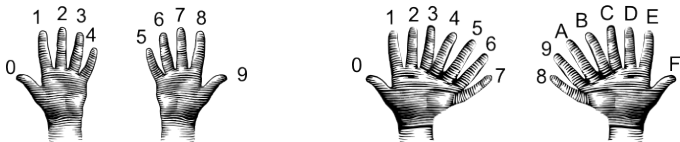
Hexadecimal to Decimal: Converting Between Bases



(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.

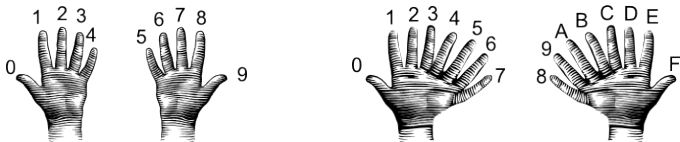
Hexadecimal to Decimal: Converting Between Bases



(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.

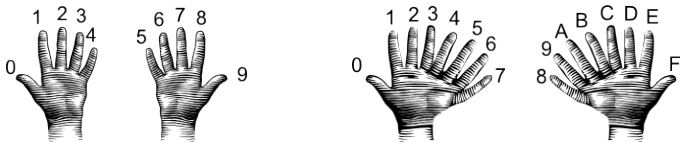
Hexadecimal to Decimal: Converting Between Bases



(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.

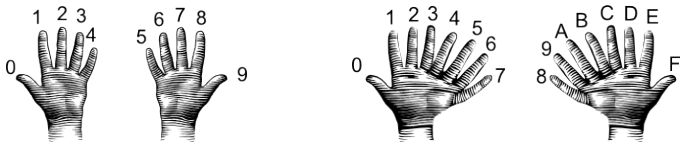
$32 + 10$ is 42.

Answer is 42.

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

9 in decimal is 9.

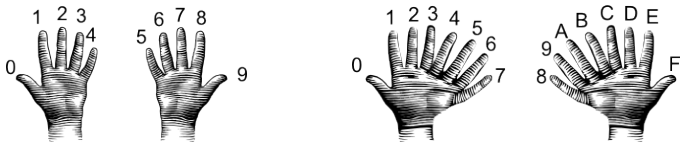
Hexadecimal to Decimal: Converting Between Bases



(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.

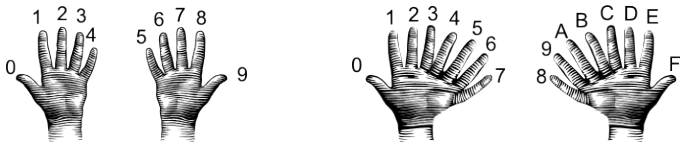
Answer is 42.

- ▶ 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.

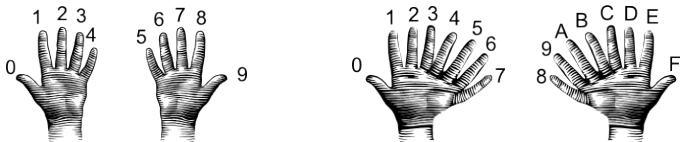
- ▶ Example: what is 99 as a decimal number?

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?

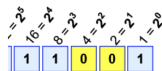
9 in decimal is 9. 9×16 is 144.

9 in decimal digits is 9

$144 + 9$ is 153.

Answer is 153.

Decimal to Binary: Converting Between Bases

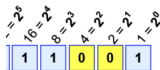


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From decimal to binary:

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

Decimal to Binary: Converting Between Bases

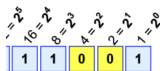


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

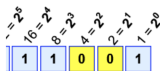


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

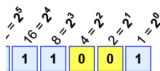


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

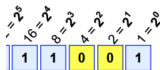


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

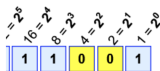


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

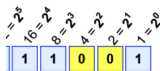


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

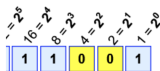


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

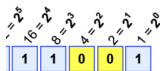


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



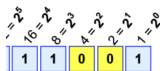
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



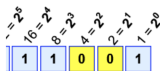
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

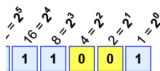
● From decimal to binary:

- ▶ 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:

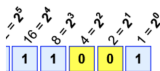
- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient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

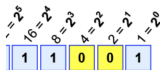
- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient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

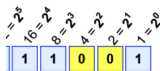
▶ Example: what is 130 in binary notation?

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:

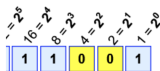
- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient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

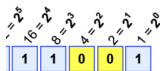
- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient 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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

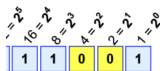
From decimal to binary:

- ▶ 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



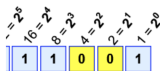
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



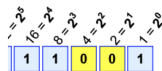
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



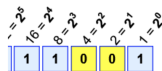
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



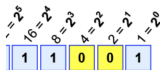
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



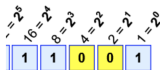
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



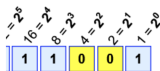
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



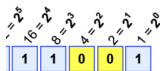
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



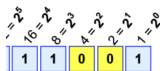
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



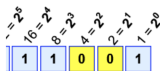
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



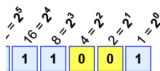
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



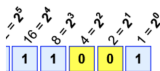
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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...

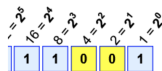
2/8 is 0 rem 2. Next digit is 0: 10000...

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...

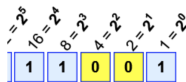
2/4 is 0 remainder 2. Next digit is 0: 100000...

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

Adding the last remainder: 10000010



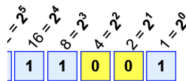
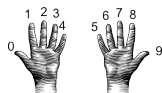
Decimal to Binary: Converting Between Bases



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

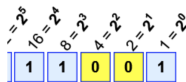
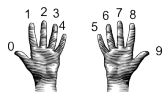


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99.

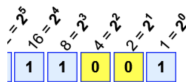
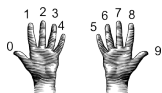
Decimal to Binary: Converting Between Bases



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



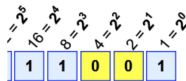
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



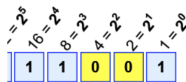
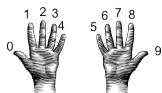
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



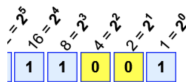
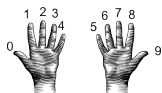
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

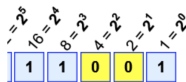
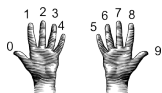
- Example: what is 99 in binary notation?

$99/128$ 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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

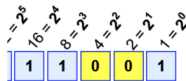
- Example: what is 99 in binary notation?

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

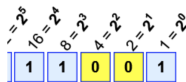
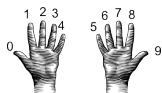
- Example: what is 99 in binary notation?

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?

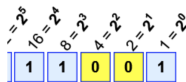
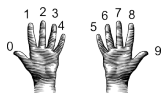
99/128 is 0 rem 99. First digit is 0: 0...

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?

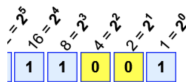
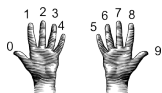
99/128 is 0 rem 99. First digit is 0: 0...

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?

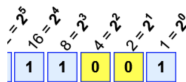
99/128 is 0 rem 99. First digit is 0: 0...

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...

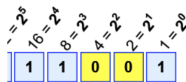
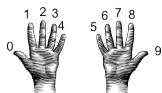
99/64 is 1 rem 35. Next digit is 1: 01...

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...

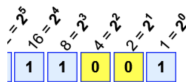
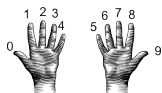
99/64 is 1 rem 35. Next digit is 1: 01...

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

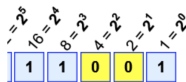


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

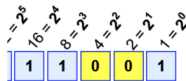


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

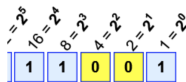
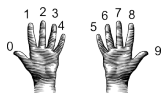
35/32 is 1 rem 3. Next digit is 1: 011...

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...

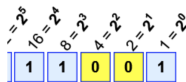
35/32 is 1 rem 3. Next digit is 1: 011...

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...

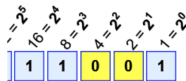
3/16 is 0 rem 3. Next digit is 0: 0110...

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...

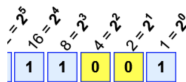
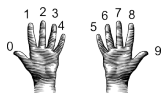
3/16 is 0 rem 3. Next digit is 0: 0110...

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

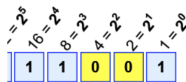
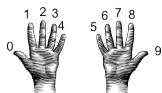


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

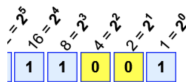
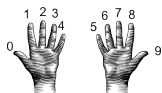
3/8 is 0 rem 3. Next digit is 0: 01100...

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



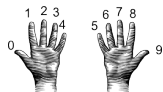
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

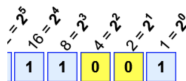
Answer is 1100011.

Binary to Decimal: Converting Between Bases



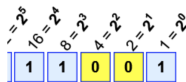
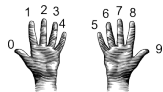
● From binary to decimal:

- ▶ Set sum = last digit.



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

Binary to Decimal: Converting Between Bases

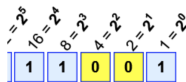
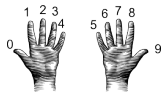


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

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

Binary to Decimal: Converting Between Bases

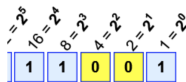
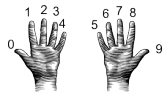


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

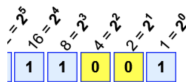


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

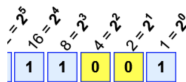
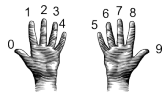


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

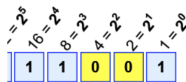
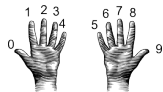


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

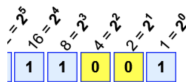
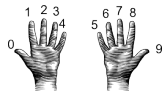


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

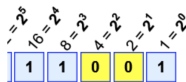
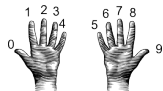


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

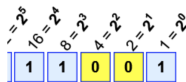
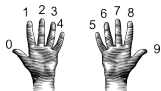


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



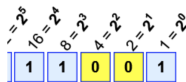
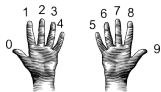
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

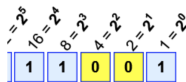
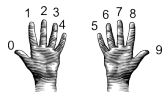
● From binary to decimal:

- ▶ 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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

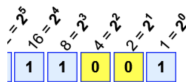
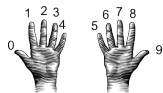
● From binary to decimal:

- ▶ 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



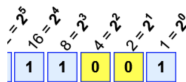
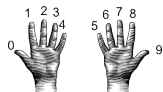
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



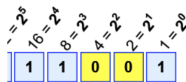
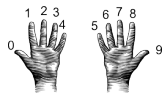
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



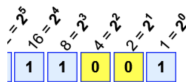
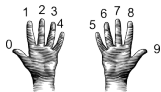
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



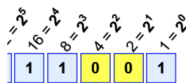
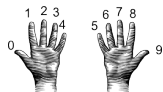
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



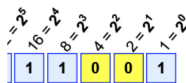
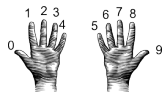
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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
 $0 \times 2 = 0$. 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:

Binary to Decimal: Converting Between Bases



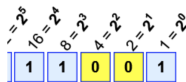
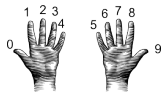
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



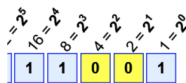
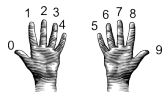
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



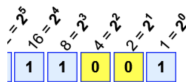
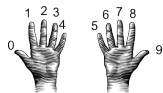
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



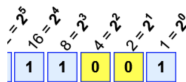
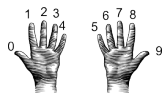
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

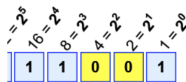
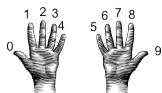


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with:

Binary to Decimal: Converting Between Bases



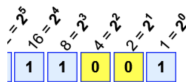
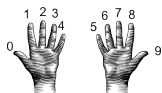
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

Binary to Decimal: Converting Between Bases



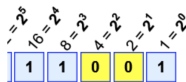
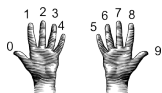
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

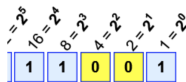
- Example: What is 10100100 in decimal?

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

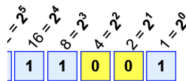
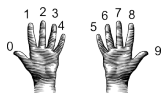
- Example: What is 10100100 in decimal?

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?

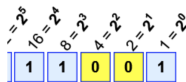
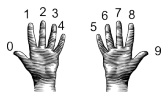
Sum starts with: 0

$0 \times 2 = 0$. 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?

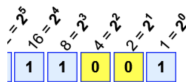
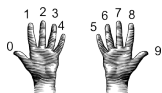
Sum starts with: 0

$0 \times 2 = 0$. 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

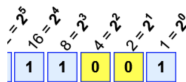
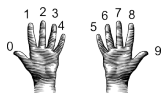


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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:

Binary to Decimal: Converting Between Bases

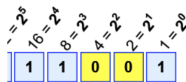
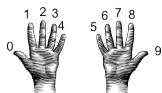


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

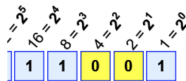
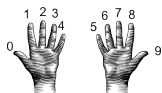
$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. 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

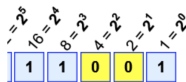
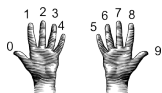


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

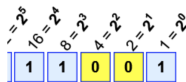


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

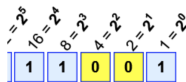
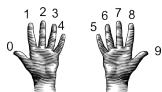


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

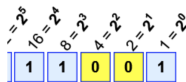
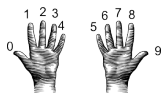


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

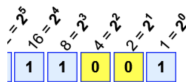
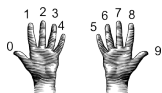


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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
$1 \times 128 = 0$. Add 128 to sum:	164

Binary to Decimal: Converting Between Bases



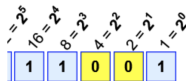
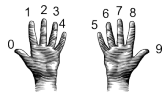
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

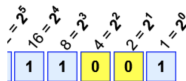
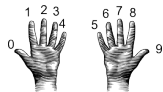
Design Challenge: Incrementers



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

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

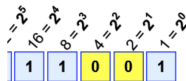
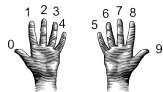
Design Challenge: Incrementers



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

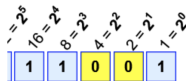
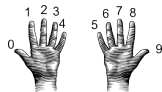


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

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

Design Challenge: Incrementers

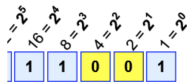
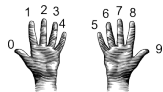


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

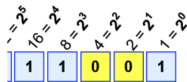
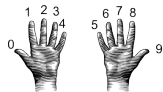


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

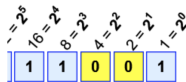
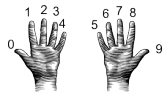


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

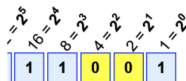
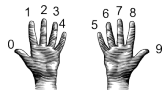


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

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.
- WeMIPS simplified machine language

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.
- WeMIPS simplified machine language
- Converting between Bases

Final Overview: Format

- The exam is 2 hours long.

Final Overview: Format

- The exam is 2 hours long.
- It is on paper. No use of computers, phones, etc. allowed.

Final Overview: Format

- The exam is 2 hours long.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.

Final Overview: Format

- The exam is 2 hours long.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.

Final Overview: Format

- The exam is 2 hours long.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Do not fold the paper; it's distracting to others taking the exam.

Final Overview: Format

- The exam is 2 hours long.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Do not fold the paper; it's distracting to others taking the exam.
 - ▶ Best if you design/write your own as it's an excellent way to study.

Final Overview: Format

- The exam is 2 hours long.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Do not fold the paper; it's distracting to others taking the exam.
 - ▶ Best if you design/write your own as it's an excellent way to study.
- The exam format:

Final Overview: Format

- The exam is 2 hours long.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Do not fold the paper; it's distracting to others taking the exam.
 - ▶ Best if you design/write your own as it's an excellent way to study.
- The exam format:
 - ▶ 10 questions, each worth 10 points.

Final Overview: Format

- The exam is 2 hours long.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Do not fold the paper; it's distracting to others taking the exam.
 - ▶ Best if you design/write your own as it's an excellent way to study.
- The exam format:
 - ▶ 10 questions, each worth 10 points.
 - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.

Final Overview: Format

- The exam is 2 hours long.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Do not fold the paper; it's distracting to others taking the exam.
 - ▶ Best if you design/write your own as it's an excellent way to study.
- The exam format:
 - ▶ 10 questions, each worth 10 points.
 - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
 - ▶ Style of questions: what does the code do? short answer, write functions, top-down design, & write complete programs.

Final Overview: Format

- The exam is 2 hours long.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Do not fold the paper; it's distracting to others taking the exam.
 - ▶ Best if you design/write your own as it's an excellent way to study.
- The exam format:
 - ▶ 10 questions, each worth 10 points.
 - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
 - ▶ Style of questions: what does the code do? short answer, write functions, top-down design, & write complete programs.
 - ▶ More on logistics next lecture.

Final Overview: Format

- The exam is 2 hours long.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Do not fold the paper; it's distracting to others taking the exam.
 - ▶ Best if you design/write your own as it's an excellent way to study.
- The exam format:
 - ▶ 10 questions, each worth 10 points.
 - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
 - ▶ Style of questions: what does the code do? short answer, write functions, top-down design, & write complete programs.
 - ▶ More on logistics next lecture.
- Past exams available on the webpage (includes answer keys).

Exam Options

Exam Times:

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

10 December 2022

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 an A 3.5" x 5" piece of paper filled with notes, programs, etc.
- While taking the exam, you may have with you pens and pencils, and your watch/stop.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open the exam until instructed to do so.

Hunter College upholds acts of academic dishonesty (e.g., plagiarism, cheating on examinations, obtaining unfair advantage, and falsification of records and official documents) as serious offenses against the values of intellectual honesty. The College is committed to enforcing 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 (required for the Honor of Academic Integrity)
Name:
Email:
Section:
Signature:

Exam Options

Exam Times:

- Regular Time: Monday, May 22 in Assembly Hall, 9-11 am.

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

10 December 2022

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 only.
- 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 upholds acts of academic dishonesty (e.g. plagiarism, cheating on examinations, obtaining unfair advantage, and falsification of records and official documents) as serious offenses against the values of intellectual honesty. The College is committed to respecting the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

At the beginning of the exam, all copies of academic dishonesty will be reported to the Office of Institutional Effectiveness and Compliance.
Name:
Signature:
Student ID:
Section:
Signature:

Exam Options

Exam Times:

- Regular Time: Monday, May 22 in Assembly Hall, 9-11 am.
- Alternate Time: Wednesday, May 17 in 1001G Hunter North, (time TBD).

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

10 December 2022

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 upholds acts of academic dishonesty (e.g. plagiarism, cheating on examinations, obtaining unfair advantage, and falsification of records and official documents) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

At the beginning of all exams of academic dishonesty will be reported to the Office of Academic Integrity.
Name:
Signature:
Grade:
Signature:

Exam Options

Exam Times:

- Regular Time: Monday, May 22 in Assembly Hall, 9-11 am.
- Alternate Time: Wednesday, May 17 in 1001G Hunter North, (time TBD).
- Survey for your exam date choice will be available next lecture.

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

10 November 2022

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.
- 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 advantage, and falsification of records and official documents) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

Identification: All cases of academic dishonesty will be reported to the Office of Institutional Effectiveness.
Name:
Height:
Weight:
Signature:

Exam Options

Exam Times:

- Regular Time: Monday, May 22 in Assembly Hall, 9-11 am.
- Alternate Time: Wednesday, May 17 in 1001G Hunter North, (time TBD).
- Survey for your exam date choice will be available next lecture.
- If you choose to take the early date, **you will not be given access to the exam on May 22, even if you miss the early exam.**

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

10 November 2022

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 3" piece of paper filled with notes, programs, etc.
- While taking the exam, you may have with you pens and pencils, and your watch.
- 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 upholds acts of academic dishonesty (i.e., plagiarism, cheating on examinations, obtaining unfair advantage, and falsification of records and official documents) as serious offenses against the values of intellectual honesty. The College is committed to respecting the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

Identification: All cases of academic dishonesty will be reported to the Office of Academic Integrity.
Name:
Height:
Weight:
Signature:

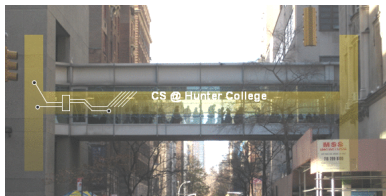
Weekly Reminders!



Before the next lecture, don't forget to:

- Work on this week's Online Lab

Weekly Reminders!



Before the next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz

Weekly Reminders!



Before the next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz
- Schedule an appointment to take the Code Review

Weekly Reminders!



Before the next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz
- Schedule an appointment to take the Code Review
- Submit this week's programming assignments

Weekly Reminders!



Before the next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz
- Schedule an appointment to take the Code Review
- Submit this week's programming assignments
- If you need help, schedule an appointment for Tutoring

Weekly Reminders!



Before the next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz
- Schedule an appointment to take the Code Review
- Submit this week's programming assignments
- If you need help, schedule an appointment for Tutoring
- Take the Lecture Preview on Blackboard

Lecture Slips & Writing Boards



- Hand your lecture slip to a UTA.
- Return writing boards as you leave.