

# CSci 127: Introduction to Computer Science



[hunter.cuny.edu/csci](https://hunter.cuny.edu/csci)

- This lecture will be recorded

# Announcements



- I will be recruiting UTAs at the end of this term. You will hear from me by email after the final exam if you earn an A (+), **if you are interested keep an eye out for my emails.**

# Announcements



- I will be recruiting UTAs at the end of this term. You will hear from me by email after the final exam if you earn an A (+), **if you are interested keep an eye out for my emails.**
- What should you do?

# Announcements



- I will be recruiting UTAs at the end of this term. You will hear from me by email after the final exam if you earn an A (+), **if you are interested keep an eye out for my emails.**
- What should you do?
  - ▶ Get an A (+)

# Announcements



- I will be recruiting UTAs at the end of this term. You will hear from me by email after the final exam if you earn an A (+), **if you are interested keep an eye out for my emails.**
- What should you do?
  - ▶ Get an A (+)
  - ▶ Be very comfortable with our labs and programming assignments

# Announcements



- I will be recruiting UTAs at the end of this term. You will hear from me by email after the final exam if you earn an A (+), **if you are interested keep an eye out for my emails.**
- What should you do?
  - ▶ Get an A (+)
  - ▶ Be very comfortable with our labs and programming assignments
  - ▶ Be able to describe a successful tutoring experience (go to tutoring!!!)

# Announcements



- I will be recruiting UTAs at the end of this term. You will hear from me by email after the final exam if you earn an A (+), **if you are interested keep an eye out for my emails.**
- What should you do?
  - ▶ Get an A (+)
  - ▶ Be very comfortable with our labs and programming assignments
  - ▶ Be able to describe a successful tutoring experience (go to tutoring!!!)
  - ▶ Previous tutoring experience helpful but not expected

# 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, 20 December, 9-11am.*

# Frequently Asked Questions

From email and tutoring.

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

*The official final is Monday, 20 December, 9-11am.*

*The early final exam (alternative date) is on Friday, 17 December, 8-10am.*

# Frequently Asked Questions

From email and tutoring.

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

*The official final is Monday, 20 December, 9-11am.*

*The early final exam (alternative date) is on Friday, 17 December, 8-10am.*

*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, 20 December, 9-11am.*

*The early final exam (alternative date) is on Friday, 17 December, 8-10am.*

*Instead of a review sheet, we have:*

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

# Frequently Asked Questions

From email and tutoring.

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

*The official final is Monday, 20 December, 9-11am.*

*The early final exam (alternative date) is on Friday, 17 December, 8-10am.*

*Instead of a review sheet, we have:*

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

# Frequently Asked Questions

From email and tutoring.

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

*The official final is Monday, 20 December, 9-11am.*

*The early final exam (alternative date) is on Friday, 17 December, 8-10am.*

*Instead of a review sheet, we have:*

- ▶ *All previous final exams (and answer keys) on the website.*
- ▶ *UTAs in drop-in tutoring happy to review concepts and old exam questions.*
- ▶ *There will be opportunity for **in-person practice** during our last meeting on 7 December.*

# Today's Topics



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

# Today's Topics



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



Predict what the code will do:

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

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

# Python Tutor

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

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

(Demo with pythonTutor)

# Design Pattern: Linear Search

- Example of **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.')
```

# 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)

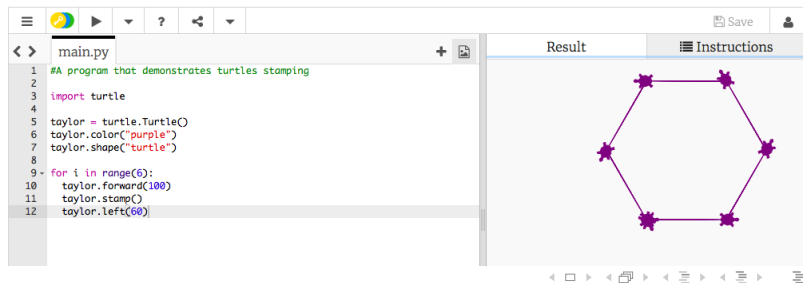
```
#This program prints: Hello, World!
```

← (this one also)

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

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

- As well as definite loops & the turtle package:



## 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.

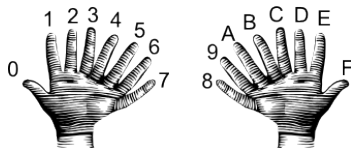
## Week 2: variables, data types, more on loops & range()

- 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)
```

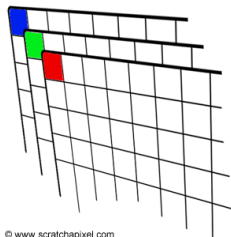
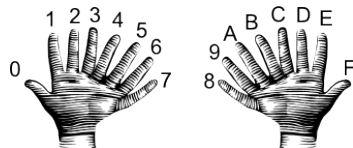
# Week 3: colors, hex, slices, numpy & images

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



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

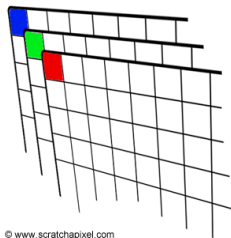
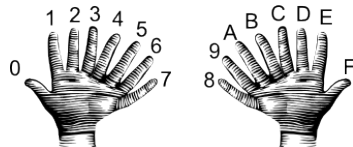
Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



© www.scratchapixel.com

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

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



© www.scratchapixel.com

```
>>> 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.")

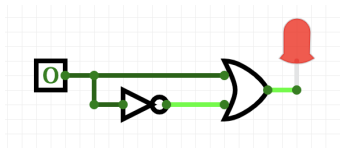
visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
    (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

# 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,,,727,7681
1771,21883,3623,,,2847,28423
1790,33131,45049,6159,1781,3827,49447
1800,40515,5740,6642,1755,4563,79215
1810,96373,40203,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419921,45468,37393,33829,1470193
1880,1164673,599495,56559,51980,38991,1911690
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018296,469042,732016,116511,5620048
1930,1867312,2560461,1079129,1265258,159346,6306446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1940101,2738275,1550849,1452177,191555,78991957
1960,1698281,2627319,1809578,1424815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7094862
1980,1428285,2230936,1801325,1168872,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8006278
2010,1548473,2504790,2230722,1385108,448730,81751123
2015,1644518,2636735,2339150,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

# Week 6: structured data, pandas, & more design

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

```
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,,30131,45049,6159,1781,3827,49447
1800,40515,5740,6642,1755,4563,79215
1810,96373,40203,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419921,45468,37393,33829,1470183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,2437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018256,469042,732016,116511,5620848
1930,1867312,2560461,1479129,1265258,159346,6306446
1940,1889924,2698295,1297634,1394711,174441,7454995
1950,1940101,2738275,1550849,1452177,191555,78931957
1960,1698281,2627319,1809578,1624815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8006278
2010,1548473,2504790,2230722,1385108,448730,81751123
2015,1644518,2636735,2339150,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6



# Week 6: structured data, pandas, & more design

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

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

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,
All population figures are consistent with present-day boundaries,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,
,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island>Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,45049,6159,1781,3827,49447
1800,40515,5740,6642,1755,4563,79215
1810,96373,40203,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419801,45468,37393,33829,1470103
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,2437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018256,469042,732016,116511,5620048
1930,1867312,2580461,1079129,1265258,159346,4590446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1940101,2738075,1500849,1451277,191555,78991957
1960,1698281,2627319,1809578,1424815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2210936,1801325,1168972,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8006278
2010,1484873,2504790,2230722,1385108,448730,81751123
2015,1644518,2636735,2339150,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

# Week 6: structured data, pandas, & more design

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

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

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

```
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
1771,21863,3623,,,2847,28423
1790,33131,4548,6159,1781,3827,49447
1800,40515,5740,6642,1755,4563,79215
1810,96373,8003,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419921,45468,37393,33829,1470193
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,2437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018256,469042,732016,116531,4620048
1930,1867312,2580461,1079129,1265258,158346,4590446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1940101,2738275,1500849,1451277,291555,78991957
1960,1698281,2627319,1809578,1624815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1801325,1168872,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1494873,2504790,2230722,1385108,448730,81751123
2015,1644518,2636735,2339150,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

# Week 6: structured data, pandas, & more design

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

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

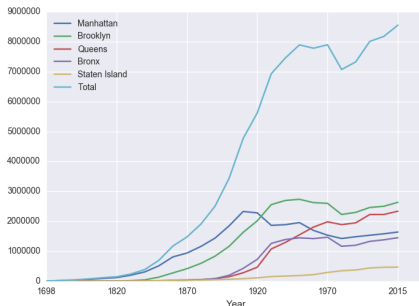
```
pop.plot(x="Year")
plt.show()
```

Source: [https://en.wikipedia.org/wiki/Demographics\\_of\\_New\\_York\\_City](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
1771,21863,3623,,2847,28423
1790,33131,4548,6159,1781,3827,49447
1800,40515,5740,6642,1755,4543,79215
1810,96373,9303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419901,45468,37393,33829,1470183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018256,469042,732016,116531,5620048
1930,1867312,2560451,1079129,1265598,159346,4906446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1940101,2738275,1500449,1452177,291559,7892957
1960,1698281,2627319,1809578,1624815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7094862
1980,1428285,2210936,1801325,1164872,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1326450,443728,8006278
2010,1484873,2504760,2230722,1385108,468730,8175133
2015,1644518,2636735,2339155,1455444,476558,8550405
```

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

- 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**.

```
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**.
- 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)
```

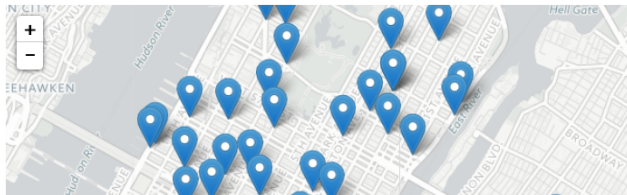
Formal Parameters

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

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

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

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

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

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

```
import turtle
import random

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

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

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

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

```
import turtle
import random
```

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

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

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

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

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

```
import turtle
import random

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

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

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

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

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

```
import turtle
import random
```

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

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

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

# Python & Circuits Review: 10 Weeks in 10 Minutes



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

# Lecture Quiz

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

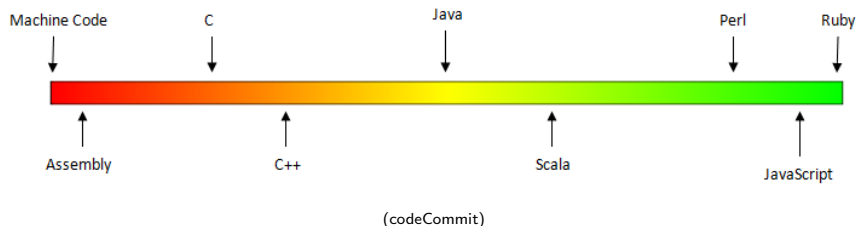


# Today's Topics



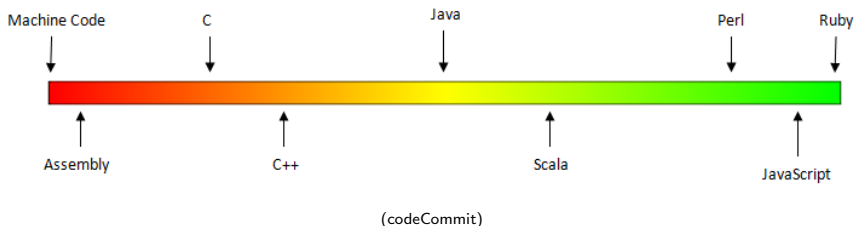
- Design Patterns: Searching
- Python Recap
- **Machine Language**
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic

# Low-Level vs. High-Level Languages



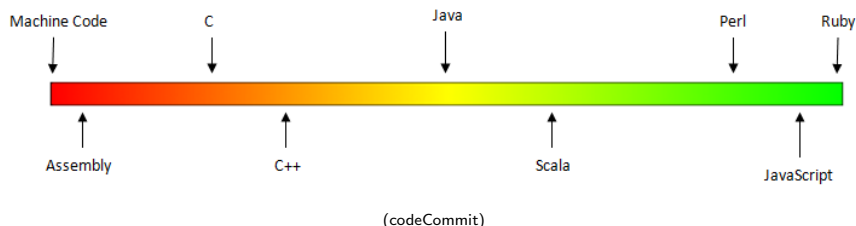
- Can view programming languages on a continuum.

# Low-Level vs. High-Level Languages



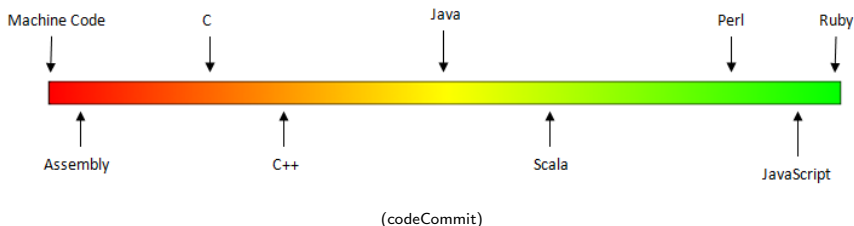
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

# Low-Level vs. High-Level Languages



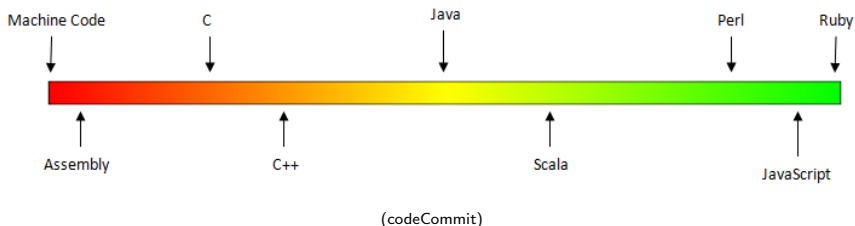
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

# Low-Level vs. High-Level Languages



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

# Low-Level vs. High-Level Languages



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

# Processing



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

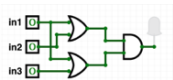


```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food * tax
    total = total + tip
    return(total)
```

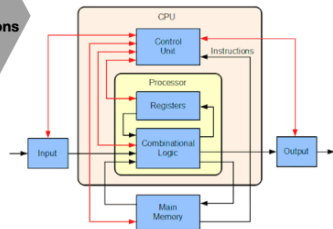
**Data  
&  
Instructions**



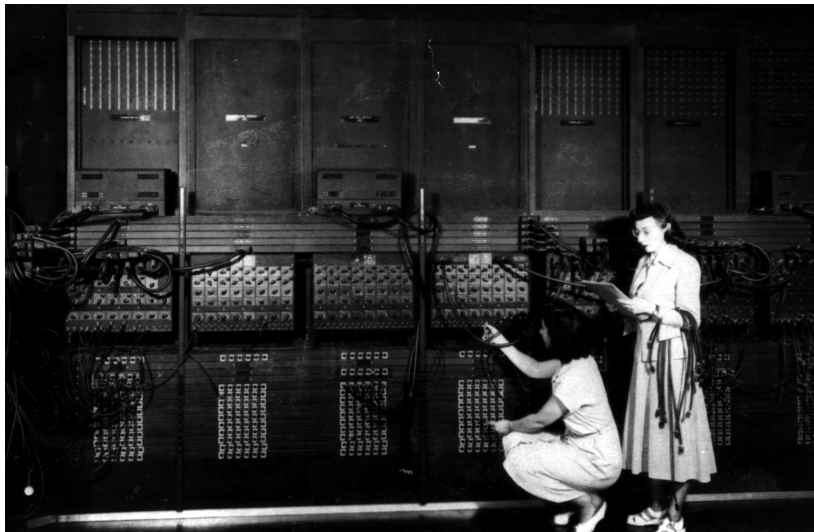
**Data  
&  
Instructions**



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



# 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 NUmxDI2C .A .X .Y SP DP DB
; 00 E012 00110000 0000 0000 0002 CFFF 0000 00
g 2000

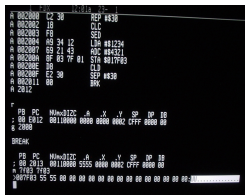
BREAK

PB PC NUmxDI2C .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.



The screenshot shows a MIPS assembly program in a debugger. The assembly code is as follows:

```
002000 c2 30      REP #30
002002 10          CLC
002003 f0          SED
002004 40 34 12     LSW #1234
002007 60 21 43     RLC #4321
002008 0f 83 7f 01  STA #17f83
00200c 00          CLD
00200f e2 30      SEP #30
002011 00          BRK
002012
```

Below the assembly code, the debugger shows the current state of the processor:

```
PC: 002012 00100000 0000 0000 0002 c7ff 0000 00
$2000
BREAK
PC: 002013 00100000 5555 0000 0002 c7ff 0000 00
n 1149 7193
000f55 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(wiki)

# Machine Language

```
002000 c2 30 REP #30
002002 10 CLC
002003 F0 SED
002004 40 34 12 LSH #1234
002007 60 21 43 RSC #04321
00200A 0F 83 7F 01 STA #017F83
00200E 30 CLD
00200F E2 30 SEP #30
002011 00 BRK
002012

P PC Mem013C: A X Y Z SP BP IB
: 00 2012 00100000 0000 0000 0002 C7FF 0000 00
S 2000

BREAK

P PC Mem013C: A X Y Z SP BP IB
: 00 2013 00100000 5555 0000 0002 C7FF 0000 00
n 1149 7193
00FFES 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



The screenshot shows the WeMIPS emulator interface. At the top, it displays assembly instructions with their addresses and hex values. Below this, the register file is shown with labels like PC, EPC, and registers A through S, each with a hex value. At the bottom, there's a memory dump showing hex values for addresses 002010 through 00201F.

```
002000 c2 30 REP #30
002002 j0 CLC
002003 f0 SED
002004 a0 34 12 LSW #1234
002007 60 21 43 RLC #04321
002008 0f 83 7f 01 STA #017f83
00200e 00 CLD
00200f e2 30 SEP #30
002011 00 BRK
002012

PC PC Mmn013C A X Y Z SP BP BB
: 00 E012 0010000 0000 0000 0002 C7FF 0000 00
$ 2000

BREAK

PC PC Mmn013C A X Y Z SP BP BB
: 00 2013 0010000 5555 0000 0002 C7FF 0000 00
n 1103 7103
002015 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(wiki)

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

# Machine Language



The screenshot shows a terminal window with assembly code and register values. The assembly code includes instructions like `REP #30`, `CLC`, `SET`, `L32 #1234`, `R0C #4321`, `STW #1789`, `CLD`, `SEP #30`, and `BRK`. The register values are displayed in a table format with columns for PC, MIPS, A, V, Y, SP, DP, and IO. The PC value is 2012, and the MIPS value is 00100000. The A register contains 0000, V contains 0000, Y contains 0000, SP contains 0000, DP contains 0000, and IO contains 0000.

(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    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

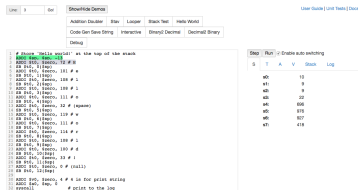
(WeMIPS)

# WeMIPS

[illegible]

(Demo with WeMIPS)

# MIPS Commands



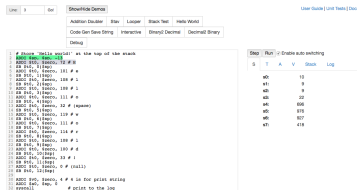
The screenshot shows the MIPS simulator interface. At the top, there are tabs for 'User Guide', 'Unit Tests', and 'Docs'. Below these are buttons for 'Show/Hide Demos', 'Assembler', 'Debugger', 'Stack Test', and 'Help Window'. A dropdown menu is open, showing options: 'Code Gen', 'Data String', 'Interactive', 'Binary', 'Decimal', and 'Decimal Binary'. The main window displays assembly code with line numbers 1 through 33. The code includes comments and instructions like 'ADDI \$t0, \$zero, 10', 'ADDI \$t1, \$zero, 9', 'ADDI \$t2, \$zero, 8', 'ADDI \$t3, \$zero, 7', 'ADDI \$t4, \$zero, 6', 'ADDI \$t5, \$zero, 5', 'ADDI \$t6, \$zero, 4', 'ADDI \$t7, \$zero, 3', 'ADDI \$t8, \$zero, 2', 'ADDI \$t9, \$zero, 1', 'ADDI \$t10, \$zero, 0', 'ADDI \$t11, \$zero, 0', 'ADDI \$t12, \$zero, 0', 'ADDI \$t13, \$zero, 0', 'ADDI \$t14, \$zero, 0', 'ADDI \$t15, \$zero, 0', 'ADDI \$t16, \$zero, 0', 'ADDI \$t17, \$zero, 0', 'ADDI \$t18, \$zero, 0', 'ADDI \$t19, \$zero, 0', 'ADDI \$t20, \$zero, 0', 'ADDI \$t21, \$zero, 0', 'ADDI \$t22, \$zero, 0', 'ADDI \$t23, \$zero, 0', 'ADDI \$t24, \$zero, 0', 'ADDI \$t25, \$zero, 0', 'ADDI \$t26, \$zero, 0', 'ADDI \$t27, \$zero, 0', 'ADDI \$t28, \$zero, 0', 'ADDI \$t29, \$zero, 0', 'ADDI \$t30, \$zero, 0', 'ADDI \$t31, \$zero, 0'. On the right, there is a 'Registers' window showing the state of registers \$0 through \$31. The 'Stop' button is highlighted, and the 'Run' button is disabled. The 'Registers' window shows the following values: \$0: 0, \$1: 0, \$2: 0, \$3: 0, \$4: 0, \$5: 0, \$6: 0, \$7: 0, \$8: 0, \$9: 0, \$10: 0, \$11: 0, \$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.



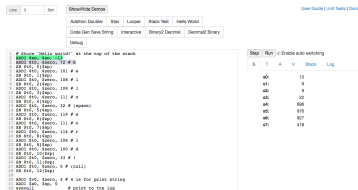


# MIPS Commands



- **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 the MIPS simulator interface. On the left, there's a text area with assembly code. On the right, there's a register window showing the state of registers \$0 through \$31.

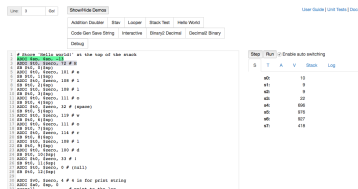
```
# MIPS "Hello world" on the top of the stack
1 addi $a0, $zero, 10 # 10
2 addi $a1, $zero, 11 # 11
3 addi $a2, $zero, 12 # 12
4 addi $a3, $zero, 13 # 13
5 addi $a4, $zero, 14 # 14
6 addi $a5, $zero, 15 # 15
7 addi $a6, $zero, 16 # 16
8 addi $a7, $zero, 17 # 17
9 addi $a8, $zero, 18 # 18
10 addi $a9,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

Register Window:

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

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

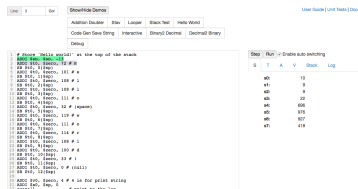
# MIPS Commands



```
1 # Please "Hello world" on the top of the screen
2 addi $v0, $zero, 10 # $v0
3 add $a0, $zero, 0 # $a0
4 li $v0, 10 # $v0
5 addi $v0, $zero, 10 # $v0
6 addi $v0, $zero, 10 # $v0
7 addi $v0, $zero, 10 # $v0
8 addi $v0, $zero, 10 # $v0
9 addi $v0, $zero, 10 # $v0
10 addi $v0, $zero, 10 # $v0
11 addi $v0, $zero, 10 # $v0
12 addi $v0, $zero, 10 # $v0
13 addi $v0, $zero, 10 # $v0
14 addi $v0, $zero, 10 # $v0
15 addi $v0, $zero, 10 # $v0
16 addi $v0, $zero, 10 # $v0
17 addi $v0, $zero, 10 # $v0
18 addi $v0, $zero, 10 # $v0
19 addi $v0, $zero, 10 # $v0
20 addi $v0, $zero, 10 # $v0
21 addi $v0, $zero, 10 # $v0
22 addi $v0, $zero, 10 # $v0
23 addi $v0, $zero, 10 # $v0
24 addi $v0, $zero, 10 # $v0
25 addi $v0, $zero, 10 # $v0
26 addi $v0, $zero, 10 # $v0
27 addi $v0, $zero, 10 # $v0
28 addi $v0, $zero, 10 # $v0
29 addi $v0, $zero, 10 # $v0
30 syscall
```

- **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.

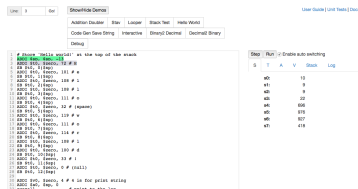
# MIPS Commands



```
1 # Please "Hello world" at the top of the window
2 addi $v0, $zero, 10 # $v0
3 add $a0, $zero, 10 # $a0
4 li $v0, 10
5 addi $v0, $zero, 10 # 1
6 li $v0, 10
7 addi $v0, $zero, 10 # 1
8 li $v0, 10
9 addi $v0, $zero, 10 # 1
10 li $v0, 10
11 addi $v0, $zero, 10 # 1
12 li $v0, 10
13 addi $v0, $zero, 10 # 1
14 li $v0, 10
15 addi $v0, $zero, 10 # 1
16 li $v0, 10
17 addi $v0, $zero, 10 # 1
18 li $v0, 10
19 addi $v0, $zero, 10 # 1
20 li $v0, 10
21 addi $v0, $zero, 10 # 1
22 li $v0, 10
23 addi $v0, $zero, 10 # 1
24 li $v0, 10
25 addi $v0, $zero, 10 # 1
26 li $v0, 10
27 addi $v0, $zero, 10 # 1
28 li $v0, 10
29 addi $v0, $zero, 10 # 1
30 li $v0, 10
```

- **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 immediate values.  
addi \$s1, \$s2, 100

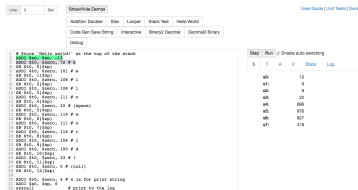
# MIPS Commands



```
1 # Please "Hello world" at the top of the window
2 addi $v0, $zero, 10 # $v0
3 add $a0, $zero
4 addi $v0, $zero, 10 # $v0
5 add $a0, $zero
6 addi $v0, $zero, 10 # $v0
7 add $a0, $zero
8 addi $v0, $zero, 10 # $v0
9 add $a0, $zero
10 addi $v0, $zero, 10 # $v0
11 add $a0, $zero
12 addi $v0, $zero, 10 # $v0
13 add $a0, $zero
14 addi $v0, $zero, 10 # $v0
15 add $a0, $zero
16 addi $v0, $zero, 10 # $v0
17 add $a0, $zero
18 addi $v0, $zero, 10 # $v0
19 add $a0, $zero
20 addi $v0, $zero, 10 # $v0
21 add $a0, $zero
22 addi $v0, $zero, 10 # $v0
23 add $a0, $zero
24 addi $v0, $zero, 10 # $v0
25 add $a0, $zero
26 addi $v0, $zero, 10 # $v0
27 add $a0, $zero
28 addi $v0, $zero, 10 # $v0
29 add $a0, $zero
30 addi $v0, $zero, 10 # $v0
```

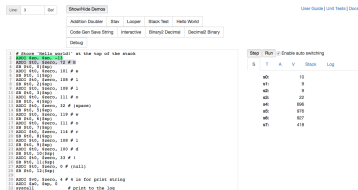
- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:  
add \$s1, \$s2, \$s3      (Basic form: OP rd, rs, rt)
- **I Instructions:** instructions that also use intermediate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

# MIPS Commands



- **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



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



# Challenge:

Line: 3    Go!

Show/Hide Demos

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

Addition Doubler

Stav

Looper

Stack Test

Hello World

Code Gen Save String

Interactive

Binary2 Decimal

Decimal2 Binary

Debug

```
1 # Store 'Hello world!' at the top of the stack
```

```
2 ADDI $sp, $sp, -13
```

```
3 ADDI $t0, $zero, 72 # H
```

```
4 SB $t0, 0($sp)
```

```
5 ADDI $t0, $zero, 101 # e
```

```
6 SB $t0, 1($sp)
```

```
7 ADDI $t0, $zero, 108 # l
```

```
8 SB $t0, 2($sp)
```

```
9 ADDI $t0, $zero, 108 # l
```

```
10 SB $t0, 3($sp)
```

```
11 ADDI $t0, $zero, 111 # o
```

```
12 SB $t0, 4($sp)
```

```
13 ADDI $t0, $zero, 32 # (space)
```

```
14 SB $t0, 5($sp)
```

```
15 ADDI $t0, $zero, 119 # w
```

```
16 SB $t0, 6($sp)
```

```
17 ADDI $t0, $zero, 111 # o
```

```
18 SB $t0, 7($sp)
```

```
19 ADDI $t0, $zero, 114 # r
```

```
20 SB $t0, 8($sp)
```

```
21 ADDI $t0, $zero, 108 # l
```

```
22 SB $t0, 9($sp)
```

```
23 ADDI $t0, $zero, 100 # d
```

```
24 SB $t0, 10($sp)
```

```
25 ADDI $t0, $zero, 33 # !
```

```
26 SB $t0, 11($sp)
```

```
27 ADDI $t0, $zero, 0 # (null)
```

```
28 SB $t0, 12($sp)
```

```
29
```

```
30 ADDI $v0, $zero, 4 # 4 is for print string
```

```
31 ADDI $a0, $sp, 0
```

```
32 syscall # print to the log
```

Step    Run    ☒ Enable auto switching

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

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

# WeMIPS

Line 3

dis

Show/Hide Demos

User Guide | Unit Tests | Docs

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

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

Debug

```
1 # Store "hello world!" at the top of the stack
2 ADDUI $a0, $zero, 32 # 8
3 R0 $a0, 0($a0)
4 ADDUI $a0, $zero, 191 # e
5 R0 $a0, 0($a0)
6 ADDUI $a0, $zero, 108 # l
7 R0 $a0, 0($a0)
8 ADDUI $a0, $zero, 108 # l
9 R0 $a0, 0($a0)
10 ADDUI $a0, $zero, 111 # o
11 R0 $a0, 0($a0)
12 ADDUI $a0, $zero, 32 # (space)
13 R0 $a0, 0($a0)
14 ADDUI $a0, $zero, 119 # w
15 R0 $a0, 0($a0)
16 ADDUI $a0, $zero, 114 # u
17 R0 $a0, 0($a0)
18 ADDUI $a0, $zero, 114 # u
19 R0 $a0, 0($a0)
20 ADDUI $a0, $zero, 108 # l
21 R0 $a0, 0($a0)
22 ADDUI $a0, $zero, 108 # l
23 R0 $a0, 0($a0)
24 ADDUI $a0, $zero, 33 # !
25 R0 $a0, 0($a0)
26 ADDUI $a0, $zero, 0 # (null)
27 R0 $a0, 0($a0)
28 ADDUI $a0, $zero, 6 # 4 in for print string
29 ADDUI $a0, $zero, 0
30 syscall # print to the log
```

Step | Run | Enable auto-switching

S	T	A	V	Stack	Log
a0				10	
a1				9	
a2				9	
a3				22	
a4				695	
a5				970	
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 screenshot shows a debugger window with two panes. The left pane displays assembly code with instructions like `movl $0, %eax`, `movl $1, %ecx`, and `jmp $0x00000000`. The right pane shows the state of registers, including `EAX`, `ECX`, and `EDX`.

# 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.



```
1  section .text
2  global _start
3  _start:
4      mov     eax, 0
5      jmp     $
6
7  section .data
8  db "A\n"
9  db "B\n"
10 db "C\n"
11 db "D\n"
12 db "E\n"
13 db "F\n"
14 db "G\n"
15 db "H\n"
16 db "I\n"
17 db "J\n"
18 db "K\n"
19 db "L\n"
20 db "M\n"
21 db "N\n"
22 db "O\n"
23 db "P\n"
24 db "Q\n"
25 db "R\n"
26 db "S\n"
27 db "T\n"
28 db "U\n"
29 db "V\n"
30 db "W\n"
31 db "X\n"
32 db "Y\n"
33 db "Z\n"
34 db "a\n"
35 db "b\n"
36 db "c\n"
37 db "d\n"
38 db "e\n"
39 db "f\n"
40 db "g\n"
41 db "h\n"
42 db "i\n"
43 db "j\n"
44 db "k\n"
45 db "l\n"
46 db "m\n"
47 db "n\n"
48 db "o\n"
49 db "p\n"
50 db "q\n"
51 db "r\n"
52 db "s\n"
53 db "t\n"
54 db "u\n"
55 db "v\n"
56 db "w\n"
57 db "x\n"
58 db "y\n"
59 db "z\n"
60 db "0\n"
61 db "1\n"
62 db "2\n"
63 db "3\n"
64 db "4\n"
65 db "5\n"
66 db "6\n"
67 db "7\n"
68 db "8\n"
69 db "9\n"
70 db " "
71 db " "
72 db " "
73 db " "
74 db " "
75 db " "
76 db " "
77 db " "
78 db " "
79 db " "
80 db " "
81 db " "
82 db " "
83 db " "
84 db " "
85 db " "
86 db " "
87 db " "
88 db " "
89 db " "
90 db " "
91 db " "
92 db " "
93 db " "
94 db " "
95 db " "
96 db " "
97 db " "
98 db " "
99 db " "
100 db " "
```

# 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.



# 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:



# 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.



# 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.



# 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

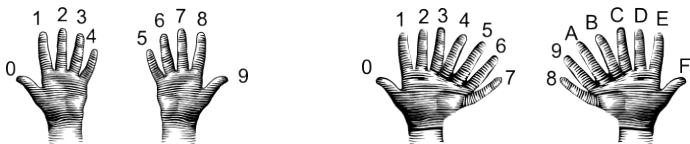
abcdefghijklmnopqrstuvwxyz

# Today's Topics



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

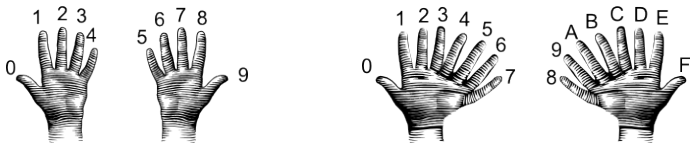
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

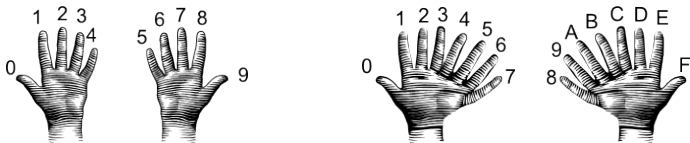
# Hexadecimal to Decimal: Converting Between Bases



(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.

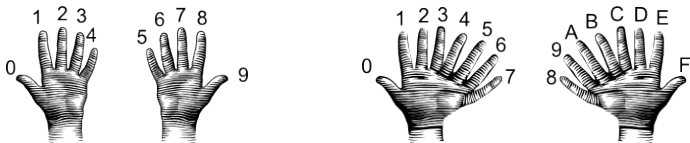
# Hexadecimal to Decimal: Converting Between Bases



(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)

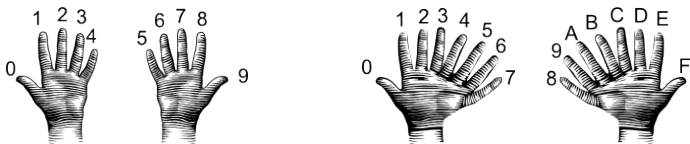
- From hexadecimal to decimal (assuming two-digit numbers):

- ▶ 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



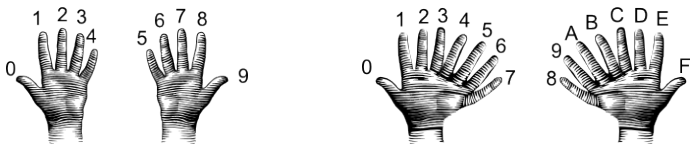
(from i-programmer.info)

- 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.  $2 \times 16$  is 32.

# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

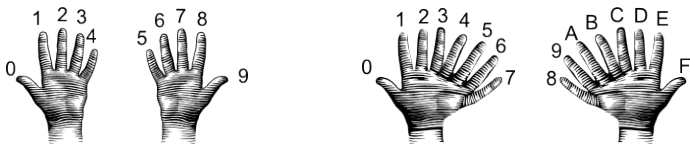
- From hexadecimal to decimal (assuming two-digit numbers):

- ▶ 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.  $2 \times 16$  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):

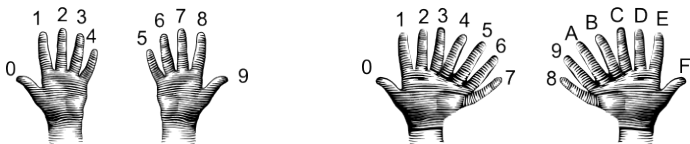
- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?

2 in decimal is 2.  $2 \times 16$  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.  $2 \times 16$  is 32.

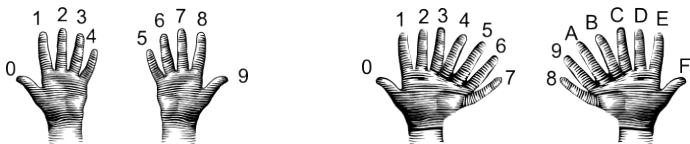
A in decimal digits is 10.

$32 + 10$  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.  $2 \times 16$  is 32.

A in decimal digits is 10.

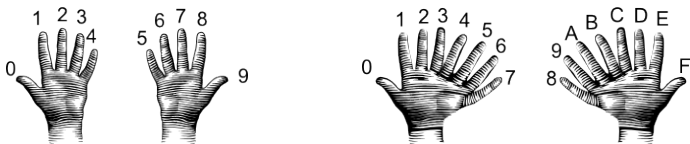
$32 + 10$  is 42.

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.  $2 \times 16$  is 32.

A in decimal digits is 10.

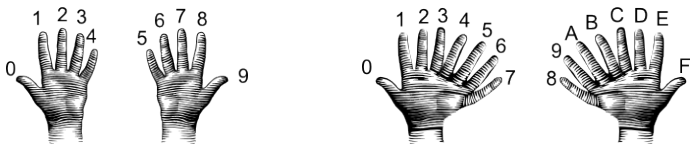
$32 + 10$  is 42.

Answer is 42.

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

9 in decimal is 9.  $9 \times 16$  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.  $2 \times 16$  is 32.

A in decimal digits is 10.

$32 + 10$  is 42.

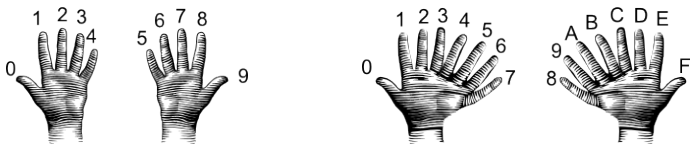
Answer is 42.

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

9 in decimal is 9.  $9 \times 16$  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.  $2 \times 16$  is 32.

A in decimal digits is 10.

$32 + 10$  is 42.

Answer is 42.

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

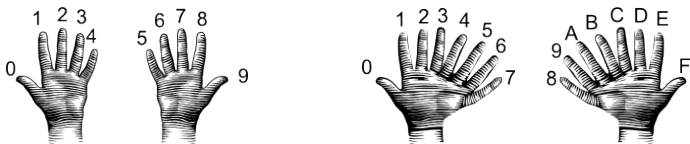
9 in decimal is 9.  $9 \times 16$  is 144.

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.  $2 \times 16$  is 32.

A in decimal digits is 10.

$32 + 10$  is 42.

Answer is 42.

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

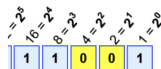
9 in decimal is 9.  $9 \times 16$  is 144.

9 in decimal digits is 9

$144 + 9$  is 153.

Answer is 153.

# Decimal to Binary: Converting Between Bases

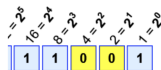


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From decimal to binary:

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

# Decimal to Binary: Converting Between Bases

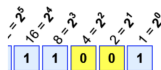


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

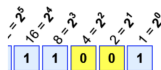


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

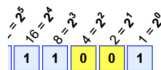


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

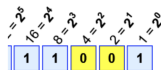


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



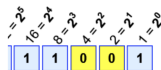
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

## 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

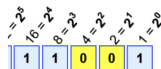


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

## 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?

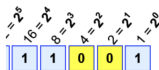
130/128 is 1 rem 2. First digit is 1: 1...

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:

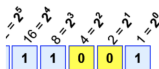
- ▶ Divide by 128 ( $= 2^7$ ). Quotient is the first digit.
- ▶ Divide remainder by 64 ( $= 2^6$ ). Quotient is the next digit.
- ▶ Divide remainder by 32 ( $= 2^5$ ). Quotient is the next digit.
- ▶ Divide remainder by 16 ( $= 2^4$ ). Quotient is the next digit.
- ▶ Divide remainder by 8 ( $= 2^3$ ).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

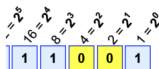
- ▶ Divide by 128 ( $= 2^7$ ). Quotient is the first digit.
- ▶ Divide remainder by 64 ( $= 2^6$ ). Quotient is the next digit.
- ▶ Divide remainder by 32 ( $= 2^5$ ). Quotient is the next digit.
- ▶ Divide remainder by 16 ( $= 2^4$ ). Quotient is the next digit.
- ▶ Divide remainder by 8 ( $= 2^3$ ).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



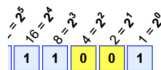
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

## ● 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



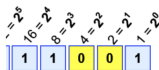
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

## 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



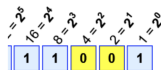
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

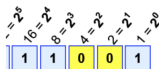
## From decimal to binary:

- ▶ 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...

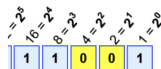
2/32 is 0 rem 2. Next digit is 0: 100...

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...

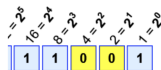
2/32 is 0 rem 2. Next digit is 0: 100...

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



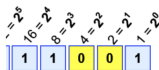
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

## ● 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...

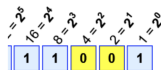
2/8 is 0 rem 2. Next digit is 0: 10000...

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...

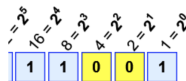
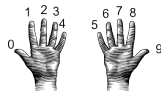
2/4 is 0 remainder 2. Next digit is 0: 100000...

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

Adding the last remainder: 10000010



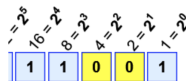
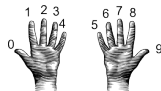
# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

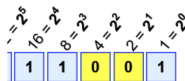
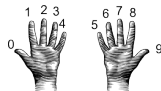
# Decimal to Binary: Converting Between Bases



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.

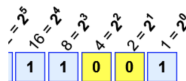
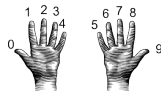
# Decimal to Binary: Converting Between Bases



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



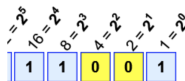
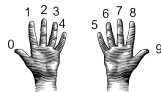
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



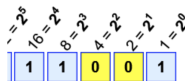
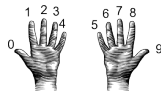
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



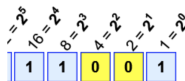
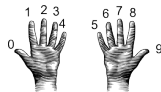
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

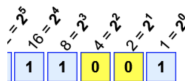
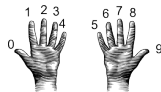
- Example: what is 99 in binary notation?

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?

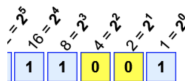
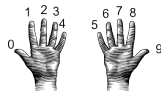
99/128 is 0 rem 99. First digit is 0: 0...

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

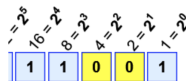
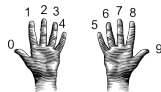
- Example: what is 99 in binary notation?

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?

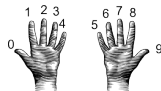
99/128 is 0 rem 99. First digit is 0: 0...

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?

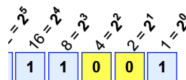
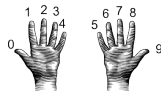
99/128 is 0 rem 99. First digit is 0: 0...

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?

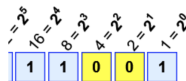
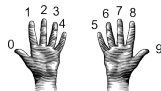
99/128 is 0 rem 99. First digit is 0: 0...

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...

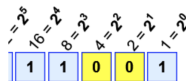
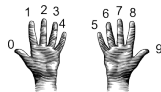
99/64 is 1 rem 35. Next digit is 1: 01...

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...

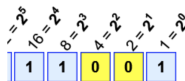
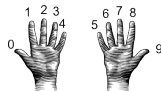
99/64 is 1 rem 35. Next digit is 1: 01...

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

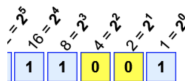
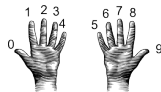


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



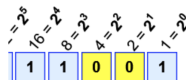
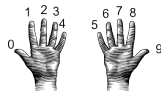
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

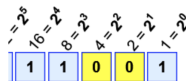
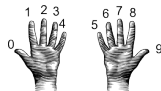


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

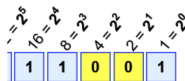
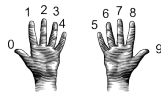
35/32 is 1 rem 3. Next digit is 1: 011...

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...

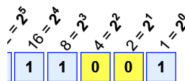
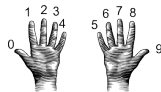
3/16 is 0 rem 3. Next digit is 0: 0110...

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...

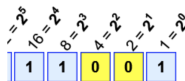
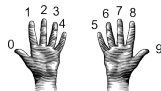
3/16 is 0 rem 3. Next digit is 0: 0110...

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...

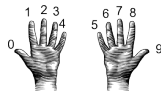
3/16 is 0 rem 3. Next digit is 0: 0110...

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

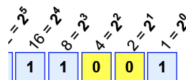
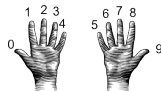


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



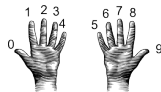
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

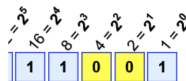
99/128 is 0 rem 99. First digit is 0:	0...
99/64 is 1 rem 35. Next digit is 1:	01...
35/32 is 1 rem 3. Next digit is 1:	011...
3/16 is 0 rem 3. Next digit is 0:	0110...
3/8 is 0 rem 3. Next digit is 0:	01100...
3/4 is 0 remainder 3. Next digit is 0:	011000...
3/2 is 1 rem 1. Next digit is 1:	0110001...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

Answer is 1100011.

# Binary to Decimal: Converting Between Bases



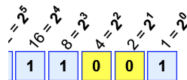
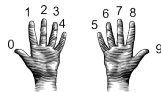
- From binary to decimal:
  - Set sum = last digit.



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



# Binary to Decimal: Converting Between Bases

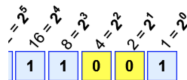
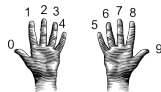


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● From binary to decimal:

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

# Binary to Decimal: Converting Between Bases

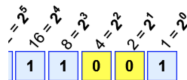
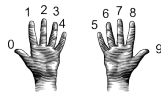


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

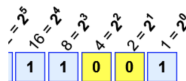
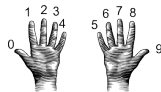


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

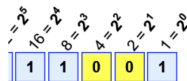
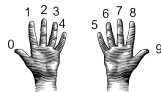


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

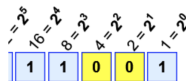
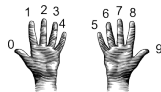


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

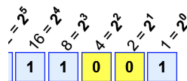
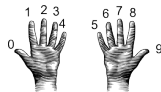


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

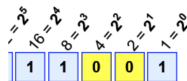
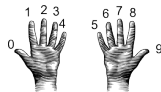


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



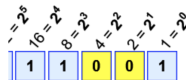
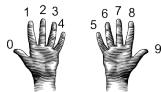
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



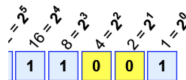
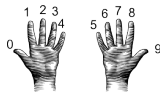
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

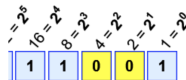
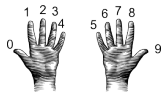
● From binary to decimal:

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

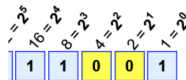
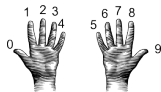
● From binary to decimal:

- ▶ 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



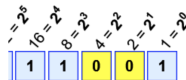
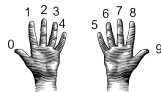
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



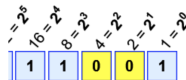
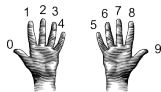
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



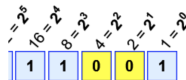
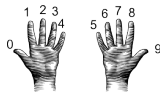
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



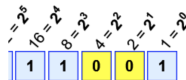
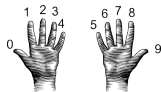
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

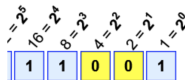
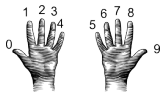
● From binary to decimal:

- ▶ 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



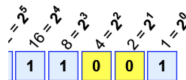
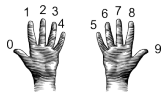
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



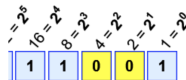
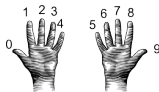
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



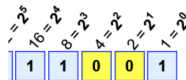
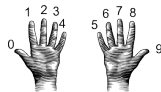
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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



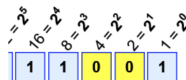
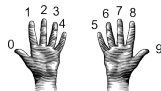
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

## ● 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

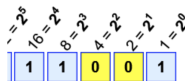
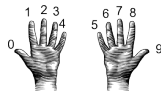


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with:

# Binary to Decimal: Converting Between Bases



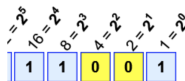
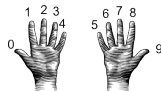
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

# Binary to Decimal: Converting Between Bases



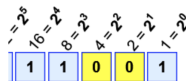
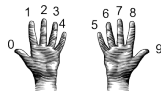
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$ . 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?

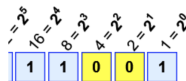
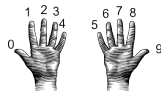
Sum starts with: 0

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

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



# Binary to Decimal: Converting Between Bases



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

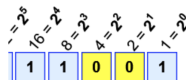
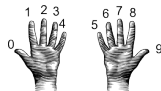
- Example: What is 10100100 in decimal?

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?

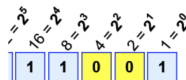
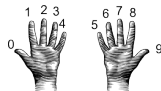
Sum starts with: 0

$0 \times 2 = 0$ . 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?

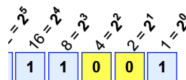
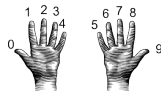
Sum starts with: 0

$0 \times 2 = 0$ . 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

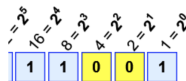
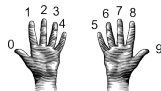
$0 \times 2 = 0$ . Add 0 to sum: 0

$1 \times 4 = 4$ . 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

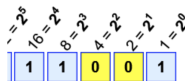
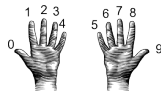
$0 \times 2 = 0$ . Add 0 to sum: 0

$1 \times 4 = 4$ . 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

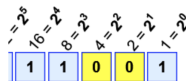
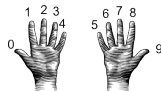
$1 \times 4 = 4$ . Add 4 to sum: 4

$0 \times 8 = 0$ . 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

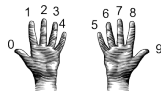


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



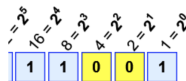
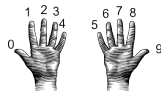
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

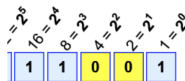
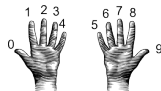


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

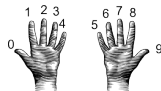


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

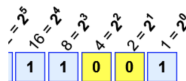
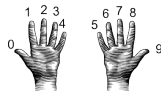


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

# Binary to Decimal: Converting Between Bases



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

- Example: What is 10100100 in decimal?

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

The answer is 164.

# 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.
- There are 4 different versions to discourage copying.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- 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.
- There are 4 different versions to discourage copying.
- 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.
- There are 4 different versions to discourage copying.
- 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.
  - ▶ No origami– it's distracting to others taking the exam.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- 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.
  - ▶ No origami– it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since excellent way to study.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- 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.
  - ▶ No origami– it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since excellent way to study.
- The exam format:



# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- 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.
  - ▶ No origami– it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
  - ▶ 10 questions, each worth 10 points.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- 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.
  - ▶ No origami– it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since 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.
- There are 4 different versions to discourage copying.
- 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.
  - ▶ No origami– it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since 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.
- There are 4 different versions to discourage copying.
- 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.
  - ▶ No origami– it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since 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.
- There are 4 different versions to discourage copying.
- 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.
  - ▶ No origami– it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since 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 webpage (includes answer keys).

# Exam Options

## Exam Times:

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

13 December 2021

### 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  $8\frac{1}{2}'' \times 11''$  piece of paper filled with notes, programs, etc.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a calculator, 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 in its algorithms, 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 upholding the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

<small>If you are not here at the time of the exam, your name will be reported to the Dean of Hunter and will result in suspension.</small>
Name:
Length:
Grade:
Signature:

# Exam Options

## Exam Times:

- Default Regular Time: Monday, 20 December, 9-11am.

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

19 December 2021

### 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 8 1/2" x 11" piece of paper filled with notes, programs, etc.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a calculator, 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 in its algorithms, cheating on examinations, cheating on assignments, and falsification of records and official documents as serious offenses against the values of intellectual honesty. The College is committed to upholding the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

<small>If permitted, have all cases of academic dishonesty will be reported to the Dean of Students and will result in sanctions.</small>
Name:
Length:
Grade:
Signature:

# Exam Options

## Exam Times:

- Default Regular Time: Monday, 20 December, 9-11am.
- Alternate Time: Friday, 17 December, 8am-10am.

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

19 December 2021

### 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 8 1/2" x 11" piece of paper filled with notes, programs, etc.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a calculator, 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 in its algorithms, 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 upholding the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

<small>If appropriate, have all cases of academic dishonesty will be reported to the Dean of Students and will result in sanctions.</small>
Name:
Length:
Grade:
Signature:



# Exam Options

## Exam Times:

- Default Regular Time: Monday, 20 December, 9-11am.
- Alternate Time: Friday, 17 December, 8am-10am.
- Accessibility Testing: Paperwork required. Must be completed on 30 November. If you have not done so already, email me no later than 23 November.

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

19 December 2021

### 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 8 1/2" x 11" piece of paper filled with notes, programs, etc.
- While taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a calculator, 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 in its algorithms, cheating on examinations, cheating on assignments, and falsification of records and official documents in various offices against the rules of institutional integrity. The College is committed to upholding the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

<small>If appropriate, have all copies of academic dishonesty will be reported to the Dean of Students and will result in sanctions.</small>
Name:
Signature:
Date:
Signature:

# Exam Options

## Exam Times:

- **Default Regular Time: Monday, 20 December, 9-11am.**
- **Alternate Time: Friday, 17 December, 8am-10am.**
- **Accessibility Testing: Paperwork required. Must be completed on 30 November. If you have not done so already, email me no later than 23 November.**
- **Survey for your exam date choice will be available next lecture. No survey answer implies you will take the exam on 20 December.**

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

13 December 2021

**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 8 1/2" x 11" piece of paper filled with notes, programs, etc.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a calculator, 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 upholding the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

all appropriate form and copies of academic dishonesty will be reported to the Dean of Students and will result in sanctions.
Name:
Signature:
Date:
Signature:

# Exam Options

## Exam Times:

- Default Regular Time: Monday, 20 December, 9-11am.
- Alternate Time: Friday, 17 December, 8am-10am.
- Accessibility Testing: Paperwork required. Must be completed on 30 November. If you have not done so already, email me no later than 23 November.
- Survey for your exam date choice will be available next lecture. **No survey answer implies you will take the exam on 20 December.**
- If you choose to take the early date, **you will not be given access to the exam on 20 December even if you miss the early exam.**

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

13 December 2021

### Exam Rules

- There are four rules. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of one 8.5" x 11" piece of paper filled with notes, programs, etc.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a calculator, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College requires acts of academic dishonesty (i.e. plagiarism, cheating on examinations, cheating on other assignments, and fabrication of research and official documents) on serious offenses against the college's intellectual integrity. The College is committed to upholding the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

signature line and name of academic dishonesty will be reported to the Dean of Students and will result in suspension
Name:
Signature:
Date:
Signature:

# Exam Options

## Exam Times:

- Default Regular Time: Monday, 20 December, 9-11am.
- Alternate Time: Friday, 17 December, 8am-10am.
- Accessibility Testing: Paperwork required. Must be completed on 30 November. If you have not done so already, email me no later than 23 November.
- Survey for your exam date choice will be available next lecture. **No survey answer implies you will take the exam on 20 December.**
- If you choose to take the early date, **you will not be given access to the exam on 20 December even if you miss the early exam.**

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

13 December 2021

### Exam Rules

- There are four rules. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of one 8.5" x 11" piece of paper filled with notes, programs, etc.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a calculator, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College requires acts of academic dishonesty (i.e. plagiarism, cheating on examinations, cheating on other assignments, and falsification of records and official documents) to receive sanctions against the student's academic standing. The College is committed to upholding the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

signature line and name of academic dishonesty will be reported to the Dean of Students and will result in sanctions.
Name:
Signature:
Date:
Signature:

# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab

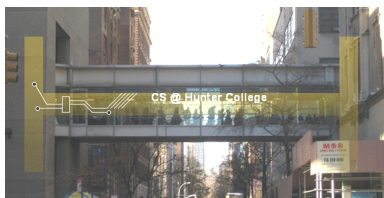
# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- [Schedule an appointment to take the Quiz](#) in lab 1001E Hunter North

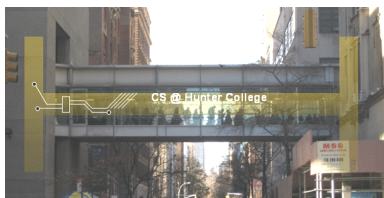
# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- [Schedule an appointment to take the Quiz](#) in lab 1001E Hunter North
- If you haven't already, [schedule an appointment to take the Code Review](#) (**one every two weeks**) in lab 1001E Hunter North

# Weekly Reminders!

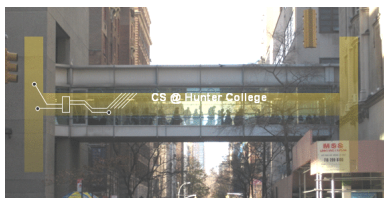


Before next lecture, don't forget to:

- Work on this week's Online Lab
- [Schedule an appointment to take the Quiz](#) in lab 1001E Hunter North
- If you haven't already, [schedule an appointment to take the Code Review](#) (**one every two weeks**) in lab 1001E Hunter North
- Submit this week's [5 programming assignments](#) (programs 51-55)



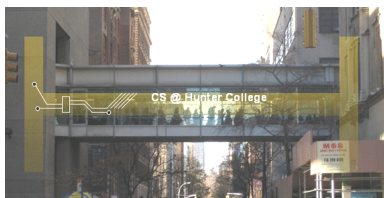
# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- [Schedule an appointment to take the Quiz](#) in lab 1001E Hunter North
- If you haven't already, [schedule an appointment to take the Code Review](#) (**one every two weeks**) in lab 1001E Hunter North
- Submit this week's [5 programming assignments](#) (programs 51-55)
- If you need help, [schedule an appointment for Tutoring](#) in lab 1001E 11am-5pm

# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- [Schedule an appointment to take the Quiz](#) in lab 1001E Hunter North
- If you haven't already, [schedule an appointment to take the Code Review](#) (**one every two weeks**) in lab 1001E Hunter North
- Submit this week's [5 programming assignments](#) (programs 51-55)
- If you need help, [schedule an appointment for Tutoring](#) in lab 1001E 11am-5pm
- Take the Lecture Preview on Blackboard on Monday (or no later than 10am on Tuesday)