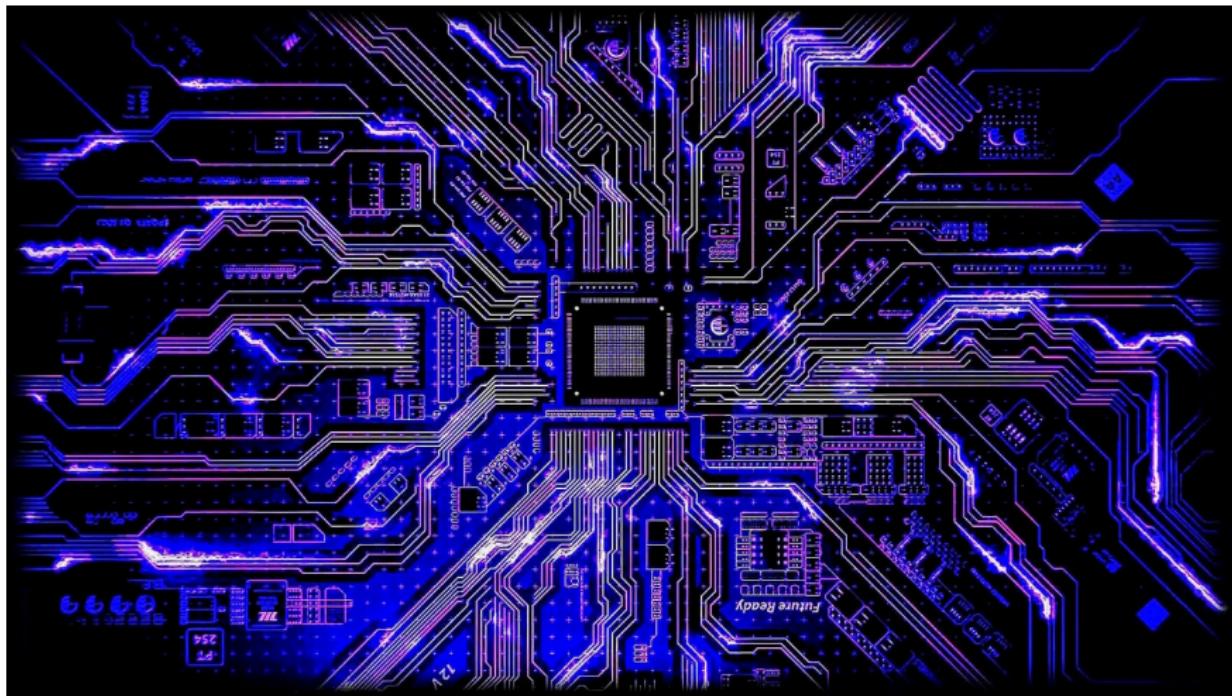


CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Frequently Asked Questions

From email and tutoring.

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

Frequently Asked Questions

From email and tutoring.

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

The official final is the last day (JULY 11) from 10am - 12pm.

Frequently Asked Questions

From email and tutoring.

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

The official final is the last day (JULY 11) from 10am - 12pm.

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 the last day (JULY 11) from 10am - 12pm.

Instead of a review sheet, we have:

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

Today's Topics



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

Today's Topics



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

Predict what the code will do:

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

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

Python Tutor

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

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

(Demo with pythonTutor)

Design Pattern: Linear Search

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

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

- Example of **linear search**.

Design Pattern: Linear Search

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

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

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

Design Pattern: Linear Search

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

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

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

Design Pattern: Linear Search

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

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

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

Today's Topics



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

Python & Circuits Review: 9 Classes in 10 Minutes



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

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

Class 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

Class 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)

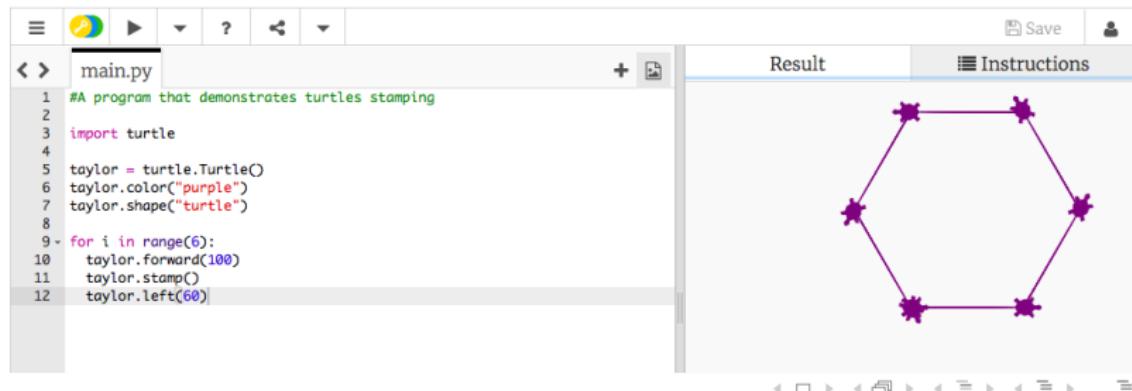
```
#This program prints: Hello, World!
```

← (this one also)

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

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

- As well as definite loops & the turtle package:



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

The screenshot shows a Python code editor with a script named 'main.py'. The code uses a for loop and the turtle module to draw a regular hexagon. The Result tab shows the drawn shape, which is a regular hexagon drawn in purple ink with black star-shaped stamps at each vertex.

Class 1: variables, data types, more on loops & range()

Class 1: variables, data types, more on loops & range()

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

Class 1: 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

Class 1: 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

Class 1: 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

Class 1: 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

Class 1: 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 1: 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.

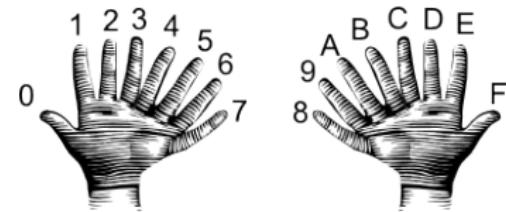
Class 1: variables, data types, more on loops & range()

- 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)
```

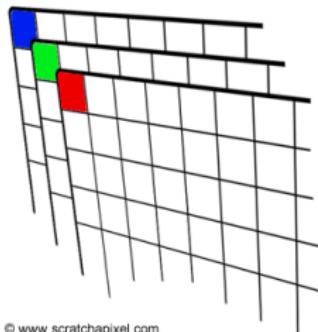
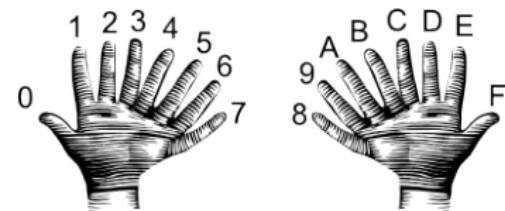
Class 2: colors, hex, slices, numpy & images

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



Class 2: colors, hex, slices, numpy & images

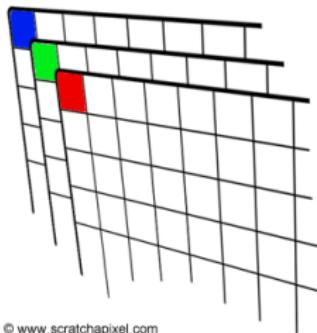
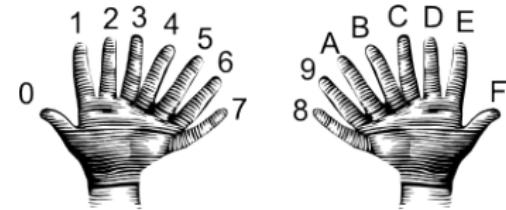
Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



© www.scratchapixel.com

Class 2: 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

Class 3: design problem (cropping images) & decisions



Class 3: design problem (cropping images) & decisions



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

Class 3: 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.

Class 3: 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.

Class 3: 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')
```

Class 4: 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!")
```

Class 4: 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	and	in2	returns:
False	and	False	False
False	and	True	False
True	and	False	False
True	and	True	True



Class 5: 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
1690,4937,2037,,727,7881
1771,21843,3623,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6442,1755,4543,75955
1810,67541,6254,6850,2030,4973,85934
1820,123704,11187,8246,2792,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312110,11013,14031,5346,10965,391114
1850,355441,128013,18951,5815,15821,5115
1860,613469,279122,32903,23593,25492,174777
1870,942292,419921,45468,37393,33029,1479103
1880,1164473,59943,5653,51980,33091,1911801
1890,1364473,70000,6353,57454,35254,2184014
1900,185093,116582,152999,200567,67621,2437202
1910,2233142,1634351,2841,430980,8569,4766803
1920,2211103,2018354,44601,44601,73201,11651,50048
1930,1867137,2203936,1797128,1797128,15821,4930446
1940,1889924,2499285,1297634,1394711,1374441,7454995
1950,1960101,2738175,1550949,1451277,191555,7891957
1960,1660101,2738175,1550949,1451277,191555,7891984
1970,1660101,2738175,1550949,1451277,191555,7891984
1980,1426285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1380789,378977,7322564
2000,1537195,2485326,2229379,1332450,419728,8080879
2010,1583873,2504705,2216722,1385108,447558,8175405
2015,1444518,2436733,2339150,1459444,474558,8056405

nycHistPop.csv

In Lab 6

Class 5: 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.....
Five census after the consolidation of the five boroughs.....

.....
Year,Borough,Brooklyn,Queens,Bronx,Staten Island,Total
1690,4937,2037,,727,7881
1771,21843,36231,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6442,1755,4543,75955
1810,67534,5940,6442,1755,4543,75934
1820,123704,11187,8246,2792,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,311510,11013,14031,5346,10965,391114
1850,355441,128911,18891,5346,10965,411115
1860,813469,279122,32903,23593,25492,174777
1870,942292,419921,45468,37393,33029,1479103
1880,1164473,59943,5653,51980,33029,1911801
1890,1385093,716582,61581,51980,33029,2141314
1900,185093,116582,152999,200567,67921,2437202
1910,2233142,1634351,28491,430980,8569,476683
1920,2211103,2018354,44671,44671,720201,11651,50048
1930,2667113,2211103,44671,44671,720201,11651,50046
1940,1889924,2498285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550949,1451277,191555,7991957
1960,1690311,2738175,1550949,1451277,191555,7981984
1970,1539231,2460711,1472701,1472701,135443,7984642
1980,1426285,2230936,1891325,1168972,252121,7071639
1990,1487536,2300664,1951598,1203789,1709777,7232564
2000,1537195,2485326,2229379,1332450,419728,8080879
2010,1583873,2504705,2229722,1385108,419728,8175133
2015,1444018,2636733,2339150,1459446,474558,8059405

nycHistPop.csv

In Lab 6

Class 5: 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  
1690,4937,2037,...,727,7881,10000  
1771,21843,36231,...,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75935  
1810,68111,6240,6842,1755,4543,83934  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,18013,14081,5348,10965,391114  
1850,355491,21800,18851,5834,11865,451115  
1860,513469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33091,1911801  
1890,1364473,72000,6534,57350,33091,2141134  
1900,1850993,116582,152999,200567,67921,2437202  
1910,2233142,1634351,28441,430980,8569,476683  
1920,2216103,2018354,44601,44601,73201,11651,500848  
1930,1867112,2081254,479128,479128,55831,4930446  
1940,1889924,2469285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7891957  
1960,1690101,2319319,1890941,1451277,191555,7891984  
1970,1539231,2460701,1471701,1471701,135443,7891984  
1980,1426285,2230936,1891325,1168972,252121,7071639  
1990,1487536,2300664,1951598,1203789,737977,7322564  
2000,1537195,2485326,2223379,1332450,419728,8080879  
2010,1583873,2504705,2217722,1385108,8175133,8175133  
2015,1444018,2636733,2339150,1459446,474558,8059405
```

nycHistPop.csv

In Lab 6

Class 5: 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,Population  
1690,4937,2017,...,727,7181  
1771,21843,36231,...,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,70000,62000,68000,1750,4543,75934  
1820,123704,11487,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,11113,14000,5348,10965,391114  
1850,355400,128000,148000,58348,135815,45115  
1860,613469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59940,5653,51980,33091,1911801  
1890,1365000,700000,650000,518600,33091,1911804  
1900,1850093,116582,152999,200567,67621,2437202  
1910,2331342,1634351,2841,430980,8569,476683  
1920,2241103,2018354,44601,44601,73201,11651,50048  
1930,1867112,1796128,1796128,1796128,1796128,4930446  
1940,1889924,2690285,1297634,1394711,1394441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690000,2319319,1809000,1809000,1809000,781984  
1970,1539231,2465701,2465701,2465701,2465443,798462  
1980,1428285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1302789,737977,7322564  
2000,1537195,2485326,2229379,1332650,419782,8080879  
2010,1583873,2504705,2277722,1385108,8175133,8175133  
2015,1444518,2640733,2339150,1459446,474558,8059405
```

nycHistPop.csv

In Lab 6

Class 5: structured data, pandas, & more design

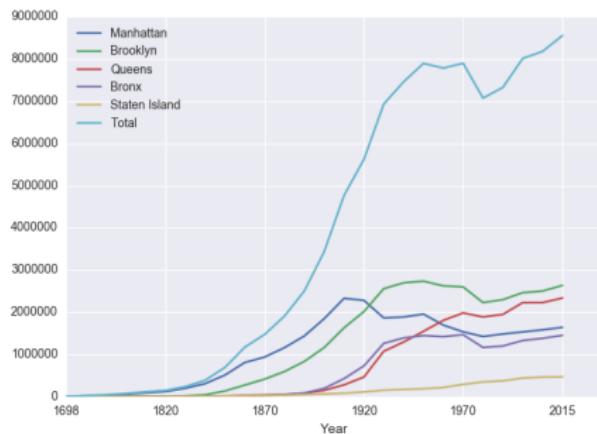
```
import matplotlib.pyplot as plt  
import pandas as pd
```

```
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,Borough,Population  
1699,Manhattan, Brooklyn, Queens, Bronx, Staten Island, Total  
1771,21843,36232,,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,70535,6211,6800,1800,4937,93734  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,18013,14000,5348,10965,391114  
1850,355441,21800,18800,5850,13000,45115  
1860,613469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33051,1911801  
1890,1380000,720000,68000,63000,58000,2310134  
1900,1850993,1165852,152999,200567,67921,2437202  
1910,2331542,1634351,2841,430980,8569,476683  
1920,2210103,2018354,44601,73201,11651,50048  
1930,2667128,2180000,579128,52545,5830,4930446  
1940,188924,2698285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550849,1451277,191555,7991957  
1960,1690000,2300000,1800000,1500000,1200000,7081984  
1970,1539231,2465000,1747301,1472701,135443,7084640  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1320789,378977,7223564  
2000,1537195,2485326,2229379,1332450,413728,8080879  
2010,1583873,2504705,2210722,1385108,413728,8175133  
2015,1444518,2636733,2339150,1459446,474558,8059405
```

nycHistPop.csv

In Lab 6



Class 6: 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()
```

Class 6: 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.

Class 6: 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()`

Class 6: 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:

Class 6: 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")`

Class 6: 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,

Class 6: 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.

Class 7: 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)
```

Class 7: 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).

Class 7: 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**.

Class 7: 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**

Class 7: 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.

Class 7: 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)
                                Actual Parameters

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.

Class 8: 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)
```

Class 9: 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)
```

Class 9: 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

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

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

Class 9: 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.

Class 9: 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`.

Class 9: 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: 9 Classes in 10 Minutes



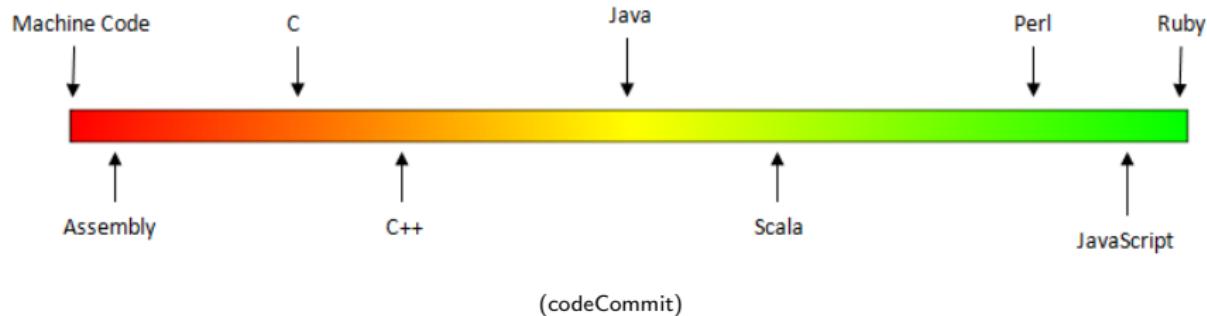
- Input/Output (I/O): `input()` and `print()`; pandas for CSV files
- Types:
 - ▶ Primitive: `int`, `float`, `bool`, `string`;
 - ▶ Container: lists (but not dictionaries/hashes or tuples)
- Objects: turtles (used but did not design our own)
- Loops: definite & indefinite
- Conditionals: if-elif-else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

Today's Topics



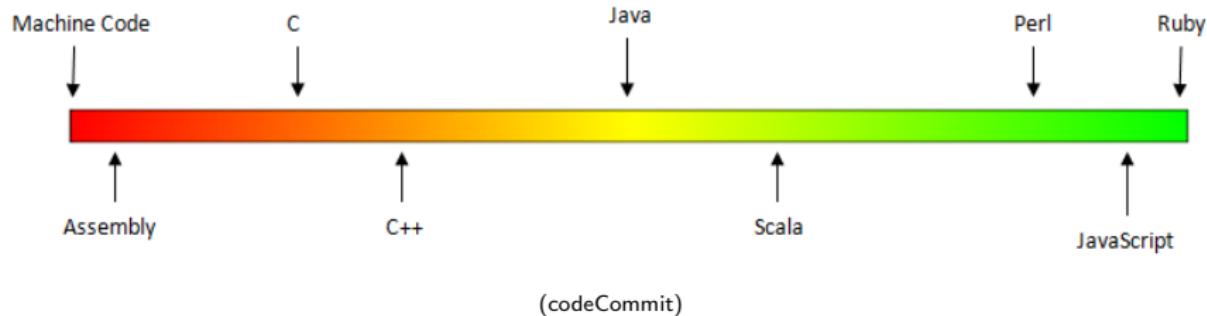
- Design Patterns: Searching
- Python Recap
- **Machine Language**
- Machine Language: Jumps & Loops
- Final Exam: Format

Low-Level vs. High-Level Languages



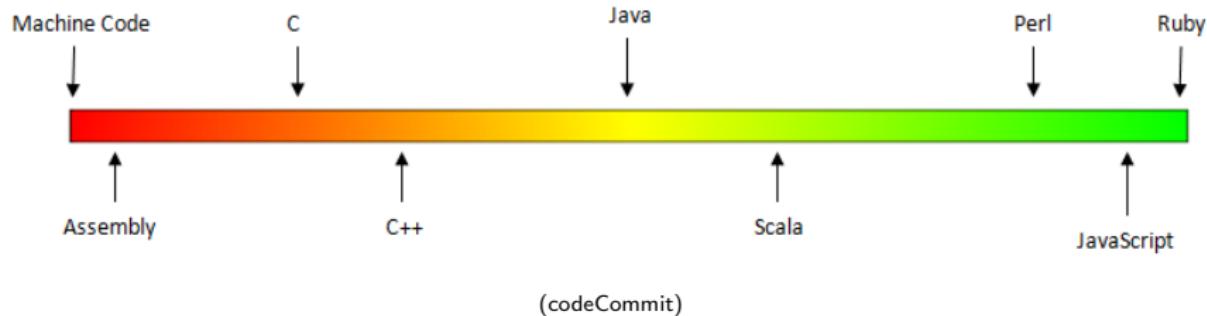
- Can view programming languages on a continuum.

Low-Level vs. High-Level Languages



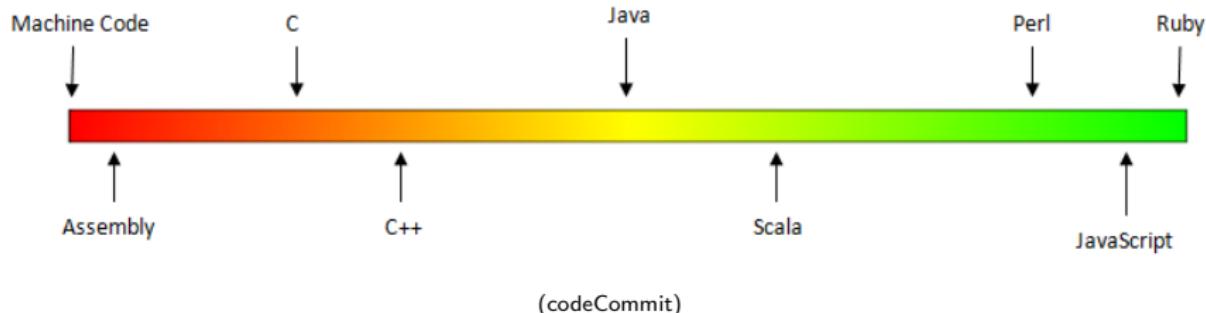
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

Low-Level vs. High-Level Languages



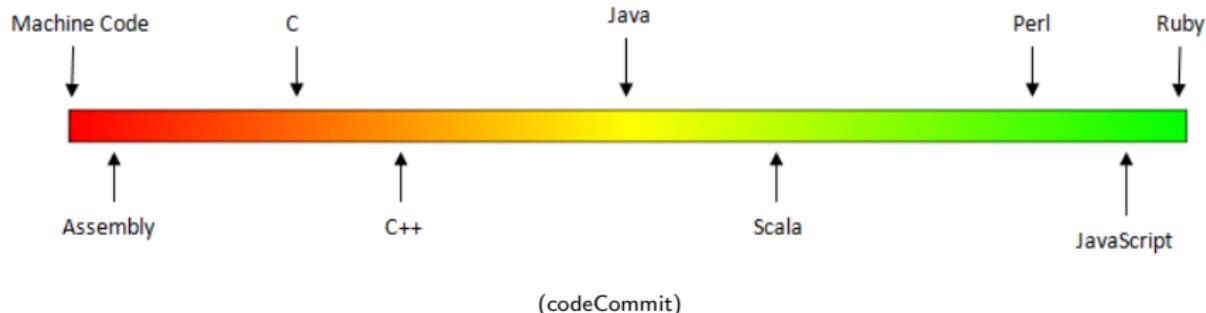
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

Low-Level vs. High-Level Languages



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

Low-Level vs. High-Level Languages



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

Processing

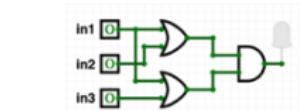
Dies ist ein Blindtext. An ihm lässt sich vieles über die Schrift ablesen, in der er gesetzt ist. Auf den ersten Blick wird der Grauwert der Schriftfläche sichtbar. Dann kann man prüfen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt.



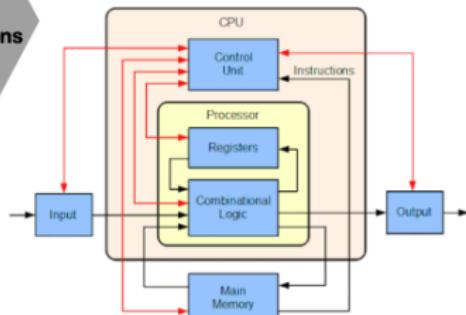
```
11 1000 1100 10011000010  
11 00 10000110101110110111  
00 1100 100001101000 101011  
00 10110010 000100001111111  
11 101101111010001111011011  
00 01010101 101101 10011000  
00 10010101011010010000001  
11110 1000000101 10011101  
01100 01011011000111010101  
0100001000000101000100 000  
0110010101 10011 1100110111  
1001001000000101110011101  
011011001 01000010000001  
111000100000000000001011100  
0110000110 101111010111001  
000001110100011001010 1110  
0110110 0110000110 1011001  
100 01 00010110100100000000  
01100110 01000000100 1001  
1001010 0011011101010110 1  
011011001100 01011001001  
111000110 0111010111011101  
01111001001 000000100001101  
0010 01000000100 1001101  
0010000001110100010111100  
00010 1 10010011 1001000000  
01110100011011110110010000  
1 1101010111001101 00101001  
01110100100101 100 10001101  
00000010110001 10110 001
```



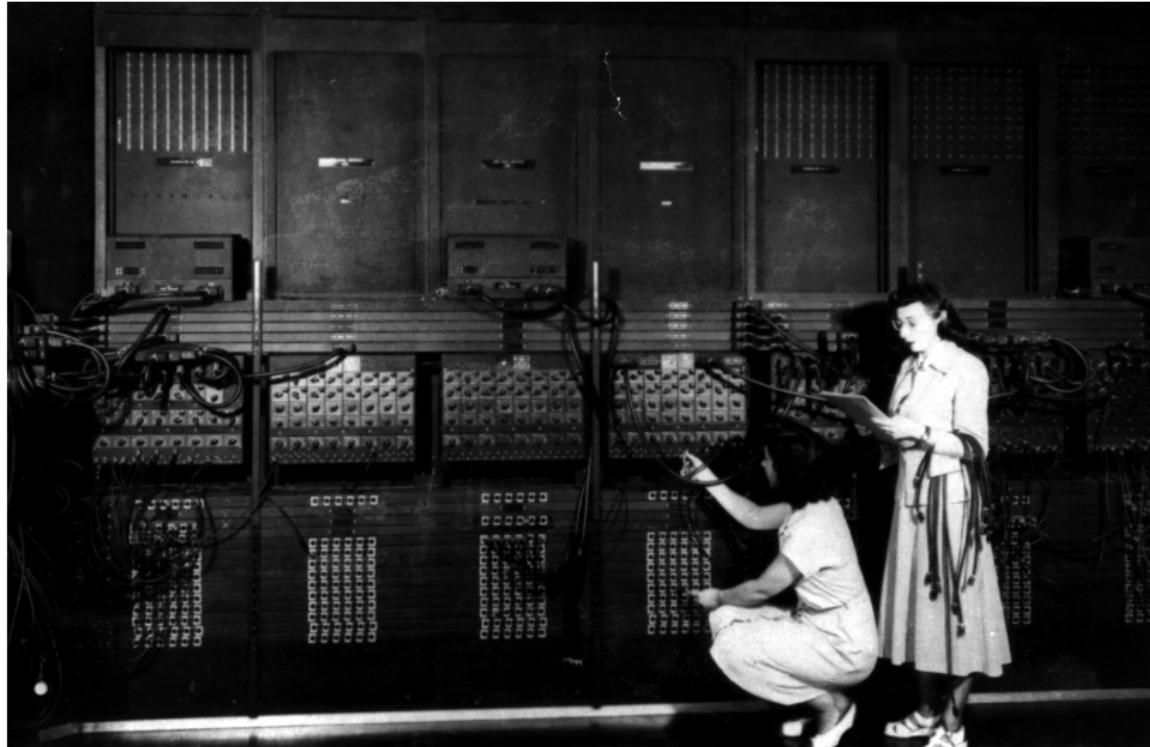
```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)
```



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



Machine Language



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

Machine Language

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

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

BREAK

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

(wiki)

Machine Language

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

(wiki)

Machine Language

- 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.

(wiki)

Machine Language

(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

(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 # i
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

(Demo with WeMIPS)

MIPS Commands

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

MIPS Commands

The screenshot shows a MIPS assembly debugger interface. At the top, there's a menu bar with 'File', 'Edit', 'Get', 'Show/Hide Demo', 'Addition', 'Subtraction', 'Multiplication', 'Stack Test', 'Hello World', 'Code Gen Base String', 'Interactive', 'Binary/Decimal', 'Decimal/Binary', and 'Debug'. Below the menu is a toolbar with 'Step', 'Run', 'Break', 'Create auto stepping', and 'Stop' buttons. To the right of the toolbar are tabs for 'S' (Registers), 'T' (Temporary Registers), 'V' (Variables), 'Stack', and 'Log'. The main area contains assembly code:

```
1 # Ensure '$main_stack' is at the top of the stack
2
3 .data
4 $main_stack: .word 11111111
5
6 .text
7 main:
8    li $t0, $main_stack
9    sb $t0, 11($t0)
10   li $t1, 11111111
11   sb $t1, 11($t0)
12   li $t2, 11111111
13   sb $t2, 11($t0)
14   li $t3, 11111111
15   sb $t3, 11($t0)
16   li $t4, 11111111
17   sb $t4, 11($t0)
18   li $t5, 11111111
19   sb $t5, 11($t0)
20   li $t6, 11111111
21   sb $t6, 11($t0)
22   li $t7, 11111111
23   sb $t7, 11($t0)
24   li $t8, 10101010
25   sb $t8, 10($t0)
26   li $t9, 11111111
27   sb $t9, 11($t0)
28   li $t10, 0
29   sb $t10, 12($t0)
30   li $t11, 11111111
31   sb $t11, 11($t0)
32
33   addi $t0, $t0, 4 # t0 = t0 + 4 for print string
34   addi $t0, $t0, 4 # t0 = t0 + 4 for print string
35
36   syscall
```

The registers section shows the following values:

Register	Value
\$t0	11
\$t1	9
\$t2	8
\$t3	22
\$t4	606
\$t5	879
\$t6	927
\$t7	411

The stack section shows the following values:

Address	Value
11	11111111
10	11111111
9	11111111
8	11111111
7	11111111
6	11111111
5	11111111
4	11111111
3	11111111
2	11111111
1	11111111
0	0

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

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

- **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

- **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

```
Line 3 Set Showchee Demos User Guide | List Tasks | Close
Additional Decoder Hex Loader Stack Test Help Window
Code Gen Save String Interactive Binary Decimal Decimal/Hexadecimal
Debug

d Shows the local code at the top of the stack
d0 00401000 00401000 $0000 0000000000000000
d1 00401000 00401000 $0000 0000000000000000
d2 00401000 00401000 $0000 0000000000000000
d3 00401000 00401000 $0000 0000000000000000
d4 00401000 00401000 $0000 0000000000000000
d5 00401000 00401000 $0000 0000000000000000
d6 00401000 00401000 $0000 0000000000000000
d7 00401000 00401000 $0000 0000000000000000
d8 00401000 00401000 $0000 0000000000000000
d9 00401000 00401000 $0000 0000000000000000
dA 00401000 00401000 $0000 0000000000000000
dB 00401000 00401000 $0000 0000000000000000
dC 00401000 00401000 $0000 0000000000000000
dD 00401000 00401000 $0000 0000000000000000
dE 00401000 00401000 $0000 0000000000000000
dF 00401000 00401000 $0000 0000000000000000
Step Run ✓ Enable auto switching
S T A V Stack Log
$0 10
$1 9
$0 8
$0 20
$0 656
$0 676
$0 807
$0 418

d ADDC $0, $0, $0, # $0 is for print string
d ADDC $0, $0, $0, # print to the log
$0x00401000 # points to the log
```

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

MIPS Commands

The screenshot shows a MIPS assembly debugger interface. At the top, there's a menu bar with 'File', 'Edit', 'Get', 'Show/Hide Demo', 'Addition', 'Subtraction', 'Multiplication', 'Division', 'Hello World', 'Code Gen', 'Save String', 'Interactive', 'Decimal Decimal', 'Decimal Binary', and 'Debug'. Below the menu is a toolbar with 'Step', 'Run', 'Break', 'Create auto watching', and 'Stop' buttons. A status bar at the bottom right says 'User Guide | Unit Tests | Docs'. The main area has tabs for 'S' (Registers), 'T' (Temporary Registers), 'V' (Variables), 'Stack', and 'Log'. The 'S' tab is selected, showing register values: \$t0=10, \$t1=9, \$t2=8, \$t3=22, \$t4=000, \$t5=819, \$t6=827, \$t7=417. The 'Log' tab displays the following assembly code:

```
1 # Shows 'Hello world' at the top of the stack
2 .text
3 .globl _start
4 .data
5 _start: .asciz "Hello world"
6 .text
7 _start: li $t0, 115902
8 sb $t0, 115902
9 li $t1, 115903
10 sb $t1, 115903
11 li $t2, 115904
12 sb $t2, 115904
13 li $t3, 115905
14 sb $t3, 115905
15 li $t4, 115906
16 sb $t4, 115906
17 li $t5, 115907
18 sb $t5, 115907
19 li $t6, 115908
20 sb $t6, 115908
21 li $t7, 115909
22 sb $t7, 115909
23 li $t8, 115910
24 sb $t8, 115910
25 li $t9, 115911
26 sb $t9, 115911
27 li $t10, 115912
28 sb $t10, 115912
29 li $t11, 115913
30 sb $t11, 115913
31 li $t12, 115914
32 sb $t12, 115914
33 li $t13, 115915
34 sb $t13, 115915
35 li $t14, 115916
36 sb $t14, 115916
37 li $t15, 115917
38 sb $t15, 115917
39 li $t16, 115918
40 sb $t16, 115918
41 li $t17, 115919
42 sb $t17, 115919
43 li $t18, 115920
44 sb $t18, 115920
45 li $t19, 115921
46 sb $t19, 115921
47 li $t20, 115922
48 sb $t20, 115922
49 li $t21, 115923
50 sb $t21, 115923
51 li $t22, 115924
52 sb $t22, 115924
53 li $t23, 115925
54 sb $t23, 115925
55 li $t24, 115926
56 sb $t24, 115926
57 li $t25, 115927
58 sb $t25, 115927
59 li $t26, 115928
60 sb $t26, 115928
61 li $t27, 115929
62 sb $t27, 115929
63 li $t28, 115930
64 sb $t28, 115930
65 li $t29, 115931
66 sb $t29, 115931
67 li $t30, 115932
68 sb $t30, 115932
69 li $t31, 115933
70 sb $t31, 115933
71 li $t32, 115934
72 sb $t32, 115934
73 li $t33, 115935
74 sb $t33, 115935
75 li $t34, 115936
76 sb $t34, 115936
77 li $t35, 115937
78 sb $t35, 115937
79 li $t36, 115938
80 sb $t36, 115938
81 li $t37, 115939
82 sb $t37, 115939
83 li $t38, 115940
84 sb $t38, 115940
85 li $t39, 115941
86 sb $t39, 115941
87 li $t40, 115942
88 sb $t40, 115942
89 li $t41, 115943
90 sb $t41, 115943
91 li $t42, 115944
92 sb $t42, 115944
93 li $t43, 115945
94 sb $t43, 115945
95 li $t44, 115946
96 sb $t44, 115946
97 li $t45, 115947
98 sb $t45, 115947
99 li $t46, 115948
100 sb $t46, 115948
101 li $t47, 115949
102 sb $t47, 115949
103 li $t48, 115950
104 sb $t48, 115950
105 li $t49, 115951
106 sb $t49, 115951
107 li $t50, 115952
108 sb $t50, 115952
109 li $t51, 115953
110 sb $t51, 115953
111 li $t52, 115954
112 sb $t52, 115954
113 li $t53, 115955
114 sb $t53, 115955
115 li $t54, 115956
116 sb $t54, 115956
117 li $t55, 115957
118 sb $t55, 115957
119 li $t56, 115958
120 sb $t56, 115958
121 li $t57, 115959
122 sb $t57, 115959
123 li $t58, 115960
124 sb $t58, 115960
125 li $t59, 115961
126 sb $t59, 115961
127 li $t60, 115962
128 sb $t60, 115962
129 li $t61, 115963
130 sb $t61, 115963
131 li $t62, 115964
132 sb $t62, 115964
133 li $t63, 115965
134 sb $t63, 115965
135 li $t64, 115966
136 sb $t64, 115966
137 li $t65, 115967
138 sb $t65, 115967
139 li $t66, 115968
140 sb $t66, 115968
141 li $t67, 115969
142 sb $t67, 115969
143 li $t68, 115970
144 sb $t68, 115970
145 li $t69, 115971
146 sb $t69, 115971
147 li $t70, 115972
148 sb $t70, 115972
149 li $t71, 115973
150 sb $t71, 115973
151 li $t72, 115974
152 sb $t72, 115974
153 li $t73, 115975
154 sb $t73, 115975
155 li $t74, 115976
156 sb $t74, 115976
157 li $t75, 115977
158 sb $t75, 115977
159 li $t76, 115978
160 sb $t76, 115978
161 li $t77, 115979
162 sb $t77, 115979
163 li $t78, 115980
164 sb $t78, 115980
165 li $t79, 115981
166 sb $t79, 115981
167 li $t80, 115982
168 sb $t80, 115982
169 li $t81, 115983
170 sb $t81, 115983
171 li $t82, 115984
172 sb $t82, 115984
173 li $t83, 115985
174 sb $t83, 115985
175 li $t84, 115986
176 sb $t84, 115986
177 li $t85, 115987
178 sb $t85, 115987
179 li $t86, 115988
180 sb $t86, 115988
181 li $t87, 115989
182 sb $t87, 115989
183 li $t88, 115990
184 sb $t88, 115990
185 li $t89, 115991
186 sb $t89, 115991
187 li $t90, 115992
188 sb $t90, 115992
189 li $t91, 115993
190 sb $t91, 115993
191 li $t92, 115994
192 sb $t92, 115994
193 li $t93, 115995
194 sb $t93, 115995
195 li $t94, 115996
196 sb $t94, 115996
197 li $t95, 115997
198 sb $t95, 115997
199 li $t96, 115998
200 sb $t96, 115998
201 li $t97, 115999
202 sb $t97, 115999
203 li $t98, 115900
204 sb $t98, 115900
205 li $t99, 115901
206 sb $t99, 115901
207 li $t100, 115902
208 sb $t100, 115902
209 li $t101, 115903
210 sb $t101, 115903
211 li $t102, 115904
212 sb $t102, 115904
213 li $t103, 115905
214 sb $t103, 115905
215 li $t104, 115906
216 sb $t104, 115906
217 li $t105, 115907
218 sb $t105, 115907
219 li $t106, 115908
220 sb $t106, 115908
221 li $t107, 115909
222 sb $t107, 115909
223 li $t108, 115910
224 sb $t108, 115910
225 li $t109, 115911
226 sb $t109, 115911
227 li $t110, 115912
228 sb $t110, 115912
229 li $t111, 115913
230 sb $t111, 115913
231 li $t112, 115914
232 sb $t112, 115914
233 li $t113, 115915
234 sb $t113, 115915
235 li $t114, 115916
236 sb $t114, 115916
237 li $t115, 115917
238 sb $t115, 115917
239 li $t116, 115918
240 sb $t116, 115918
241 li $t117, 115919
242 sb $t117, 115919
243 li $t118, 115920
244 sb $t118, 115920
245 li $t119, 115921
246 sb $t119, 115921
247 li $t120, 115922
248 sb $t120, 115922
249 li $t121, 115923
250 sb $t121, 115923
251 li $t122, 115924
252 sb $t122, 115924
253 li $t123, 115925
254 sb $t123, 115925
255 li $t124, 115926
256 sb $t124, 115926
257 li $t125, 115927
258 sb $t125, 115927
259 li $t126, 115928
260 sb $t126, 115928
261 li $t127, 115929
262 sb $t127, 115929
263 li $t128, 115930
264 sb $t128, 115930
265 li $t129, 115931
266 sb $t129, 115931
267 li $t130, 115932
268 sb $t130, 115932
269 li $t131, 115933
270 sb $t131, 115933
271 li $t132, 115934
272 sb $t132, 115934
273 li $t133, 115935
274 sb $t133, 115935
275 li $t134, 115936
276 sb $t134, 115936
277 li $t135, 115937
278 sb $t135, 115937
279 li $t136, 115938
280 sb $t136, 115938
281 li $t137, 115939
282 sb $t137, 115939
283 li $t138, 115940
284 sb $t138, 115940
285 li $t139, 115941
286 sb $t139, 115941
287 li $t140, 115942
288 sb $t140, 115942
289 li $t141, 115943
290 sb $t141, 115943
291 li $t142, 115944
292 sb $t142, 115944
293 li $t143, 115945
294 sb $t143, 115945
295 li $t144, 115946
296 sb $t144, 115946
297 li $t145, 115947
298 sb $t145, 115947
299 li $t146, 115948
300 sb $t146, 115948
301 li $t147, 115949
302 sb $t147, 115949
303 li $t148, 115950
304 sb $t148, 115950
305 li $t149, 115951
306 sb $t149, 115951
307 li $t150, 115952
308 sb $t150, 115952
309 li $t151, 115953
310 sb $t151, 115953
311 li $t152, 115954
312 sb $t152, 115954
313 li $t153, 115955
314 sb $t153, 115955
315 li $t154, 115956
316 sb $t154, 115956
317 li $t155, 115957
318 sb $t155, 115957
319 li $t156, 115958
320 sb $t156, 115958
321 li $t157, 115959
322 sb $t157, 115959
323 li $t158, 115960
324 sb $t158, 115960
325 li $t159, 115961
326 sb $t159, 115961
327 li $t160, 115962
328 sb $t160, 115962
329 li $t161, 115963
330 sb $t161, 115963
331 li $t162, 115964
332 sb $t162, 115964
333 li $t163, 115965
334 sb $t163, 115965
335 li $t164, 115966
336 sb $t164, 115966
337 li $t165, 115967
338 sb $t165, 115967
339 li $t166, 115968
340 sb $t166, 115968
341 li $t167, 115969
342 sb $t167, 115969
343 li $t168, 115970
344 sb $t168, 115970
345 li $t169, 115971
346 sb $t169, 115971
347 li $t170, 115972
348 sb $t170, 115972
349 li $t171, 115973
350 sb $t171, 115973
351 li $t172, 115974
352 sb $t172, 115974
353 li $t173, 115975
354 sb $t173, 115975
355 li $t174, 115976
356 sb $t174, 115976
357 li $t175, 115977
358 sb $t175, 115977
359 li $t176, 115978
360 sb $t176, 115978
361 li $t177, 115979
362 sb $t177, 115979
363 li $t178, 115980
364 sb $t178, 115980
365 li $t179, 115981
366 sb $t179, 115981
367 li $t180, 115982
368 sb $t180, 115982
369 li $t181, 115983
370 sb $t181, 115983
371 li $t182, 115984
372 sb $t182, 115984
373 li $t183, 115985
374 sb $t183, 115985
375 li $t184, 115986
376 sb $t184, 115986
377 li $t185, 115987
378 sb $t185, 115987
379 li $t186, 115988
380 sb $t186, 115988
381 li $t187, 115989
382 sb $t187, 115989
383 li $t188, 115990
384 sb $t188, 115990
385 li $t189, 115991
386 sb $t189, 115991
387 li $t190, 115992
388 sb $t190, 115992
389 li $t191, 115993
390 sb $t191, 115993
391 li $t192, 115994
392 sb $t192, 115994
393 li $t193, 115995
394 sb $t193, 115995
395 li $t194, 115996
396 sb $t194, 115996
397 li $t195, 115997
398 sb $t195, 115997
399 li $t196, 115998
400 sb $t196, 115998
401 li $t197, 115999
402 sb $t197, 115999
403 li $t198, 115900
404 sb $t198, 115900
405 li $t199, 115901
406 sb $t199, 115901
407 li $t200, 115902
408 sb $t200, 115902
409 li $t201, 115903
410 sb $t201, 115903
411 li $t202, 115904
412 sb $t202, 115904
413 li $t203, 115905
414 sb $t203, 115905
415 li $t204, 115906
416 sb $t204, 115906
417 li $t205, 115907
418 sb $t205, 115907
419 li $t206, 115908
420 sb $t206, 115908
421 li $t207, 115909
422 sb $t207, 115909
423 li $t208, 115910
424 sb $t208, 115910
425 li $t209, 115911
426 sb $t209, 115911
427 li $t210, 115912
428 sb $t210, 115912
429 li $t211, 115913
430 sb $t211, 115913
431 li $t212, 115914
432 sb $t212, 115914
433 li $t213, 115915
434 sb $t213, 115915
435 li $t214, 115916
436 sb $t214, 115916
437 li $t215, 115917
438 sb $t215, 115917
439 li $t216, 115918
440 sb $t216, 115918
441 li $t217, 115919
442 sb $t217, 115919
443 li $t218, 115920
444 sb $t218, 115920
445 li $t219, 115921
446 sb $t219, 115921
447 li $t220, 115922
448 sb $t220, 115922
449 li $t221, 115923
450 sb $t221, 115923
451 li $t222, 115924
452 sb $t222, 115924
453 li $t223, 115925
454 sb $t223, 115925
455 li $t224, 115926
456 sb $t224, 115926
457 li $t225, 115927
458 sb $t225, 115927
459 li $t226, 115928
460 sb $t226, 115928
461 li $t227, 115929
462 sb $t227, 115929
463 li $t228, 115930
464 sb $t228, 115930
465 li $t229, 115931
466 sb $t229, 115931
467 li $t230, 115932
468 sb $t230, 115932
469 li $t231, 115933
470 sb $t231, 115933
471 li $t232, 115934
472 sb $t232, 115934
473 li $t233, 115935
474 sb $t233, 115935
475 li $t234, 115936
476 sb $t234, 115936
477 li $t235, 115937
478 sb $t235, 115937
479 li $t236, 115938
480 sb $t236, 115938
481 li $t237, 115939
482 sb $t237, 115939
483 li $t238, 115940
484 sb $t238, 115940
485 li $t239, 115941
486 sb $t239, 115941
487 li $t240, 115942
488 sb $t240, 115942
489 li $t241, 115943
490 sb $t241, 115943
491 li $t242, 115944
492 sb $t242, 115944
493 li $t243, 115945
494 sb $t243, 115945
495 li $t244, 115946
496 sb $t244, 115946
497 li $t245, 115947
498 sb $t245, 115947
499 li $t246, 115948
500 sb $t246, 115948
501 li $t247, 115949
502 sb $t247, 115949
503 li $t248, 115950
504 sb $t248, 115950
505 li $t249, 115951
506 sb $t249, 115951
507 li $t250, 115952
508 sb $t250, 115952
509 li $t251, 115953
510 sb $t251, 115953
511 li $t252, 115954
512 sb $t252, 115954
513 li $t253, 115955
514 sb $t253, 115955
515 li $t254, 115956
516 sb $t254, 115956
517 li $t255, 115957
518 sb $t255, 115957
519 li $t256, 115958
520 sb $t256, 115958
521 li $t257, 115959
522 sb $t257, 115959
523 li $t258, 115960
524 sb $t258, 115960
525 li $t259, 115961
526 sb $t259, 115961
527 li $t260, 115962
528 sb $t260, 115962
529 li $t261, 115963
530 sb $t261, 115963
531 li $t262, 115964
532 sb $t262, 115964
533 li $t263, 115965
534 sb $t263, 115965
535 li $t264, 115966
536 sb $t264, 115966
537 li $t265, 115967
538 sb $t265, 115967
539 li $t266, 115968
540 sb $t266, 115968
541 li $t267, 115969
542 sb $t267, 115969
543 li $t268, 115970
544 sb $t268, 115970
545 li $t269, 115971
546 sb $t269, 115971
547 li $t270, 115972
548 sb $t270, 115972
549 li $t271, 115973
550 sb $t271, 115973
551 li $t272, 115974
552 sb $t272, 115974
553 li $t273, 115975
554 sb $t273, 115975
555 li $t274, 115976
556 sb $t274, 115976
557 li $t275, 115977
558 sb $t275, 115977
559 li $t276, 115978
560 sb $t276, 115978
561 li $t277, 115979
562 sb $t277, 115979
563 li $t278, 115980
564 sb $t278, 115980
565 li $t279, 115981
566 sb $t279, 115981
567 li $t280, 115982
568 sb $t280, 115982
569 li $t281, 115983
570 sb $t281, 115983
571 li $t282, 115984
572 sb $t282, 115984
573 li $t283, 115985
574 sb $t283, 115985
575 li $t284, 115986
576 sb $t284, 115986
577 li $t285, 115987
578 sb $t285, 115987
579 li $t286, 115988
580 sb $t286, 115988
581 li $t287, 115989
582 sb $t287, 115989
583 li $t288, 115990
584 sb $t288, 115990
585 li $t289, 115991
586 sb $t289, 115991
587 li $t290, 115992
588 sb $t290, 115992
589 li $t291, 115993
590 sb $t291, 115993
591 li $t292, 115994
592 sb $t292, 115994
593 li $t293, 115995
594 sb $t293, 115995
595 li $t294, 115996
596 sb $t294, 115996
597 li $t295, 115997
598 sb $t295, 115997
599 li $t296, 115998
600 sb $t296, 115998
601 li $t297, 115999
602 sb $t297, 115999
603 li $t298, 115900
604 sb $t298, 115900
605 li $t299, 115901
606 sb $t299, 115901
607 li $t300, 115902
608 sb $t300, 115902
609 li $t301, 115903
610 sb $t301, 115903
611 li $t302, 115904
612 sb $t302, 115904
613 li $t303, 115905
614 sb $t303, 115905
615 li $t304, 115906
616 sb $t304, 115906
617 li $t305, 115907
618 sb $t305, 115907
619 li $t306, 115908
620 sb $t306, 115908
621 li $t307, 115909
622 sb $t307, 115909
623 li $t308, 115910
624 sb $t308, 115910
625 li $t309, 115911
626 sb $t309, 115911
627 li $t310, 115912
628 sb $t310, 115912
629 li $t311, 115913
630 sb $t311, 115913
631 li $t312, 115914
632 sb $t312, 115914
633 li $t313, 115915
634 sb $t313, 115915
635 li $t314, 115916
636 sb $t314, 115916
637 li $t315, 115917
638 sb $t315, 115917
639 li $t316, 115918
640 sb $t316, 115918
641 li $t317, 115919
642 sb $t317, 115919
643 li $t318, 115920
644 sb $t318, 115920
645 li $t319, 115921
646 sb $t319, 115921
647 li $t320, 115922
648 sb $t320, 115922
649 li $t321, 115923
650 sb $t321, 115923
651 li $t322, 115924
652 sb $t322, 115924
653 li $t323, 115925
654 sb $t323, 115925
655 li $t324, 115926
656 sb $t324, 115926
657 li $t325, 115927
658 sb $t325, 115927
659 li $t326, 115928
660 sb $t326, 115928
661 li $t327, 115929
662 sb $t327, 115929
663 li $t328, 115930
664 sb $t328, 115930
665 li $t329, 115931
666 sb $t329, 115931
667 li $t330, 115932
668 sb $t330, 115932
669 li $t331, 115933
670 sb $t331, 115933
671 li $t332, 115934
672 sb $t332, 115934
673 li $t333, 115935
674 sb $t333, 115935
675 li $t334, 115936
676 sb $t334, 115936
677 li $t335, 115937
678 sb $t335, 115937
679 li $t336, 115938
680 sb $t336, 115938
681 li $t337, 115939
682 sb $t337, 115939
683 li $t338, 115940
684 sb $t338, 115940
685 li $t339, 115941
686 sb $t339, 115941
687 li $t340, 115942
688 sb $t340, 115942
689 li $t341, 115943
690 sb $t341, 115943
691 li $t342, 115944
692 sb $t
```

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

```
Line 3 Set Showchee Demos User Guide | List Tasks | Close
Additional Decoder Hex Loader Stack Test Help Window
Code Gen Save String Interactive Binary Decimal Decimal/Hexadecimal
Debug

d Shows the local code at the top of the stack
d0 00401000 00401000 $0000 0000000000000000
d1 00401000 00401000 $0000 0000000000000000
d2 00401000 00401000 $0000 0000000000000000
d3 00401000 00401000 $0000 0000000000000000
d4 00401000 00401000 $0000 0000000000000000
d5 00401000 00401000 $0000 0000000000000000
d6 00401000 00401000 $0000 0000000000000000
d7 00401000 00401000 $0000 0000000000000000
d8 00401000 00401000 $0000 0000000000000000
d9 00401000 00401000 $0000 0000000000000000
dA 00401000 00401000 $0000 0000000000000000
dB 00401000 00401000 $0000 0000000000000000
dC 00401000 00401000 $0000 0000000000000000
dD 00401000 00401000 $0000 0000000000000000
dE 00401000 00401000 $0000 0000000000000000
dF 00401000 00401000 $0000 0000000000000000
Step Run ✓ Enable auto switching
S T A V Stack Log

d0 10
d1 9
d2 8
d3 20
d4 656
d5 676
d6 807
d7 418

d Shows the local code at the top of the stack
d0 00401000 00401000 $0000 0000000000000000
d1 00401000 00401000 $0000 0000000000000000
d2 00401000 00401000 $0000 0000000000000000
d3 00401000 00401000 $0000 0000000000000000
d4 00401000 00401000 $0000 0000000000000000
d5 00401000 00401000 $0000 0000000000000000
d6 00401000 00401000 $0000 0000000000000000
d7 00401000 00401000 $0000 0000000000000000
d8 00401000 00401000 $0000 0000000000000000
d9 00401000 00401000 $0000 0000000000000000
dA 00401000 00401000 $0000 0000000000000000
dB 00401000 00401000 $0000 0000000000000000
dC 00401000 00401000 $0000 0000000000000000
dD 00401000 00401000 $0000 0000000000000000
dE 00401000 00401000 $0000 0000000000000000
dF 00401000 00401000 $0000 0000000000000000
ADDQ $10, %esp;    # esp = 0 for prior saving
movl %esp, %ebp;   # points to the top
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.
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      # print to the log
32 syscall
```

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

(Demo with WeMIPS)

Today's Topics



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

Loops & Jumps in Machine Language

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



A screenshot of a debugger interface. On the left, the assembly code window shows a series of instructions starting with `LDI R0, #1000`. On the right, the registers window shows the state of various寄存器 (Registers) including R0 through R15, along with memory and stack pointers.

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.



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 ADDI $sp, $sp, -27      # Set up stack
2 ADDI $s3, $zero, 1       # Store 1 in a register
3 ADDI $t0, $zero, 97      # Set $t0 at 97 (a)
4 ADDI $s2, $zero, 26      # Use to test when you reach 26
5 SETUP: SB $t0, 0($sp)    # Next letter in $t0
6 ADDI $sp, $sp, 1         # Increment the stack
7 SUB $s2, $s2, $s3        # Decrease the counter by 1
8 ADDI $t0, $t0, 1         # Increment the letter
9 BEQ $s2, $zero, DONE     # Jump to done if $s2 == 0
10 J SETUP
11 J 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



Recap



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

Recap



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

Today's Topics



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

Final Overview: Administration

- The exam will be administered in class, on paper.

Final Overview: Administration

- The exam will be administered in class, on paper.
- Please arrive 15 minutes earlier than the start time so there is no rush and chaos
- The exam format:

Final Overview: Administration

- The exam will be administered in class, on paper.
- Please arrive 15 minutes earlier than the start time so there is no rush and chaos
- The exam format:
 - ▶ Questions will correspond to the parts of old exams, the types of topics/concepts will be same but obviously different questions

Final Overview: Administration

- The exam will be administered in class, on paper.
- Please arrive 15 minutes earlier than the start time so there is no rush and chaos
- The exam format:
 - ▶ Questions will correspond to the parts of old exams, the types of topics/concepts will be same but obviously different questions
 - ▶ Questions are variations on the programming assignments, lab exercises, and lecture design challenges.

Final Overview: Administration

- The exam will be administered in class, on paper.
- Please arrive 15 minutes earlier than the start time so there is no rush and chaos
- The exam format:
 - ▶ Questions will correspond to the parts of old exams, the types of topics/concepts will be same but obviously different questions
 - ▶ Questions are variations on the programming assignments, lab exercises, and lecture design challenges.

Final Overview: Format

- You are allowed to prepare and bring in 1 piece of **8.5" x 11"** paper.

Final Overview: Format

- You are allowed to prepare and bring in 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.

Final Overview: Format

- You are allowed to prepare and bring in 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Best if you design/write yours since excellent way to study.

Final Overview: Format

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

Final Overview: Format

- You are allowed to prepare and bring in 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Best if you design/write yours since excellent way to study.
 - ▶ Avoid scrambling through web searches and waste time during the exam.
- Past exams available on webpage (includes answer keys).

Class Reminders!



Before next lecture, don't forget to:

- Review this class's Lecture and Lab 11!

Class Reminders!



Before next lecture, don't forget to:

- Review this class's Lecture and Lab 11!
- Batch 9 is due TOMORROW 6pm!!!