

# CSci 127: Introduction to Computer Science



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

# Announcements

- Thanksgiving Break starts in 9 days.



# Announcements



- Thanksgiving Break starts in 9 days.
- No CUNY classes:  
*Thursday-Saturday, 28-31 November.*

# Announcements



- Thanksgiving Break starts in 9 days.
- No CUNY classes:  
*Thursday-Saturday, 28-31 November.*
- In response to wrap-up requests, additional challenges today with while loops and binary & hexadecimal numbers.

# Today's Topics



- Python Recap
- Design Patterns: Searching
- 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

# In Pairs or Triples:

Predict what the code will do:

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

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

# Python Tutor

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

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

(Demo with pythonTutor)

# Design Pattern: Linear Search

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

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

- Example of linear search.

# Design Pattern: Linear Search

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

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

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

# Design Pattern: Linear Search

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

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

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

# Design Pattern: Linear Search

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

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

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

# Today's Topics



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

# Python & Circuits Review: 10 Weeks in 10 Minutes



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

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

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

- Introduced comments & print():

```
#Name: Thomas Hunter
```

← These lines are comments

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

← (for us, not computer to read)

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

← (this one also)

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

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

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

- Introduced comments & print():

```
#Name: Thomas Hunter
```

← These lines are comments

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

← (for us, not computer to read)

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

← (this one also)

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

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

- As well as definite loops & the turtle package:

The screenshot shows a Python code editor interface. The left pane displays the code file `main.py` with the following content:

```
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 right pane has two tabs: `Result` and `Instructions`. The `Result` tab shows the output of the program, which is a regular hexagon drawn in purple. Each vertex of the hexagon has a small purple star-like stamp.

# 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.

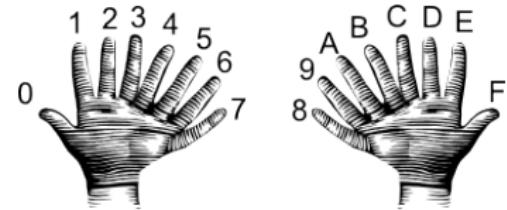
## Week 2: variables, data types, more on loops & range()

- 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(x)  
12  
13 for c in "ABCD":  
14     print(c)
```

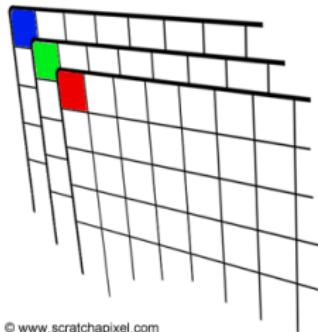
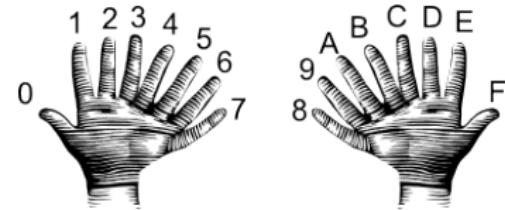
# Week 3: colors, hex, slices, numpy & images

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



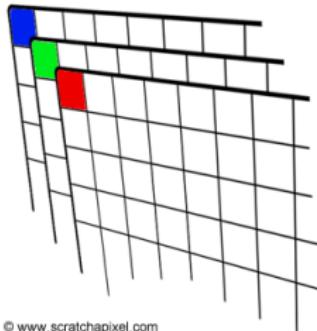
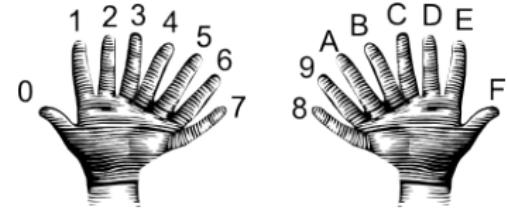
# Week 3: colors, hex, slices, numpy & images

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



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

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



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

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

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

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

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

# Week 4: design problem (cropping images) & decisions



# Week 4: design problem (cropping images) & decisions



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

# Week 4: design problem (cropping images) & decisions



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

# Week 4: design problem (cropping images) & decisions



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

## Week 4: design problem (cropping images) & decisions

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

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

# Week 5: logical operators, truth tables & logical circuits

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

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

# Week 5: logical operators, truth tables & logical circuits

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

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

in1	and	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](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  
1890,1,937,2037,727,788,2942  
1870,33131,4549,6159,1781,3827,49447  
1860,60515,5740,6442,1755,4543,75955  
1850,55349,5254,5911,1575,4397,74934  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,21613,14049,5348,10965,391114  
1850,35549,21891,18959,5348,10965,391115  
1860,813469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59945,5653,51989,33021,1911801  
1890,1367711,70001,6548,5634,41861,2186134  
1900,185093,116582,152999,200567,67621,2437202  
1910,223342,1634351,264041,430980,8569,4766803  
1920,2211103,2018354,44603,44603,73201,11651,50048  
1930,1867112,1797128,2230936,1891325,158245,4930446  
1940,1889924,2698285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550849,1451277,191555,7991957  
1960,1690101,2738175,1890949,1451277,191555,7981984  
1970,1539231,2460705,1471705,1471705,135443,7981984  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1203789,378977,7322564  
2000,1537195,2485326,2229379,1332650,419782,8080879  
2010,1583873,2504705,2216722,1385108,4175133,8175133  
2015,1444518,2646733,2339159,1454446,474558,8056405

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](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  
1890,4937,2037,,727,7881  
1871,21843,3623,,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,63545,5540,6242,1755,4543,75934  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,19113,14049,5346,10965,391114  
1850,35549,21890,18895,10212,10965,49115  
1860,813469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59945,5653,51980,33029,1911803  
1890,1367111,70000,65000,51980,33029,2141134  
1900,185093,116582,152999,200567,67621,2437202  
1910,2233142,1634351,284041,430980,8569,4766803  
1920,2210103,2018354,446071,446071,732013,1165100,50048  
1930,1667111,1796128,1796128,1796128,1796128,4930446  
1940,1889924,2498285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690101,2319319,1890949,1451277,191555,7981984  
1970,1539231,2465701,1471701,1471701,135443,7981984  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1203789,378977,7322564  
2000,1537195,2485326,2229379,1332450,419728,8080879  
2010,1583873,2504705,2277722,1385108,4175133,8175133  
2015,1444018,2646733,2339150,1459444,474558,8059405

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  
1690,4937,2037,727,788,104  
1771,21843,3623,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,67541,6240,6840,2000,4543,89734  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,19013,14084,5346,10965,391114  
1850,355441,21890,18891,5346,10965,415115  
1860,613469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33091,1911801  
1890,1367111,70000,65000,58000,33091,2347134  
1900,1850593,116582,152999,200567,67921,2437202  
1910,2233142,1634351,2841,430980,8569,476683  
1920,22101103,2018354,44600,72021,11651,50048  
1930,1867111,1579128,1579128,1579128,1579128,4930446  
1940,1889924,2498285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690101,2738175,1550949,1451277,191555,7981984  
1970,1539231,2465705,1472705,1271705,135443,7984846  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1203789,378977,7322564  
2000,1537195,2485326,2223379,1332450,419782,8080879  
2010,1583873,2504705,2271722,1385108,419782,8175133  
2015,1444018,2546733,2339150,1459446,474558,8059405
```

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.....  
Five census after the consolidation of the five boroughs.....  
.....  
Year,Population  
1690,203,2037,...,727,7181  
1771,21843,36243,...,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,70000,60000,60000,1750,4543,75934  
1820,123704,11487,8246,2792,6135,152056  
1830,20589,20535,9049,3023,7082,242278  
1840,31510,21013,14000,5346,10965,391114  
1850,35549,21890,14895,5346,10965,391115  
1860,813469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33021,1911801  
1890,1385700,718700,68000,51980,33021,1911804  
1900,1850993,116582,152999,200567,67621,2437202  
1910,223342,1634351,28441,430980,8569,476683  
1920,2246103,2018354,44601,44601,73201,11671,50048  
1930,2667128,2079128,44601,44601,73201,11671,50046  
1940,1889924,2498285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690000,2738175,1550949,1451277,191555,7991984  
1970,1539231,2465701,1472701,1472701,135443,7991946  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1302789,378977,7322564  
2000,1537195,2485326,2229379,1332650,419782,8080879  
2010,1583873,2504705,2272722,1385108,474558,8175133  
2015,1444018,2646733,2339150,1459446,474558,8175405
```

nycHistPop.csv

In Lab 6

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

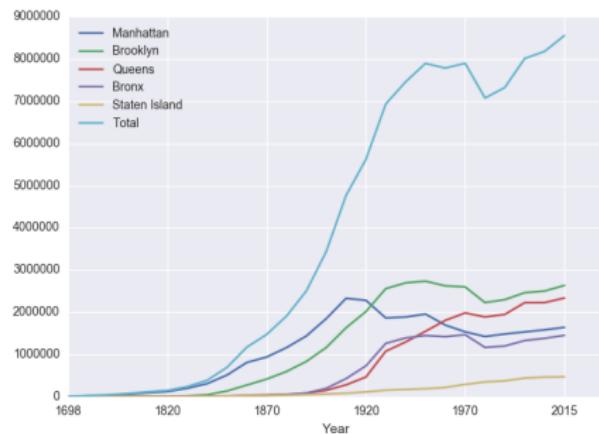
```
import matplotlib.pyplot as plt  
import pandas as pd
```

```
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,Borough,Population  
1698,Manhattan,2037,727,7181  
1771,21843,36231,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,70000,6350,7000,1800,5000,89734  
1820,123704,11187,8246,2792,6135,152056  
1830,202889,20535,9049,3023,7082,242278  
1840,312110,12013,14000,5348,10965,391114  
1850,435441,128000,18500,65000,15000,41515  
1860,613469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59940,5653,51980,33051,1911801  
1890,1384473,72000,60000,58000,35000,2000000  
1900,1850093,116582,152999,200567,67921,2437202  
1910,233142,1634351,2841,430980,8589,476683  
1920,2281103,2018354,44600,73201,11650,50048  
1930,2667103,2486000,47473,72051,15840,580446  
1940,1889924,2698285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550849,1451277,191555,7091957  
1960,1690000,27319,1809000,1400000,1800000,7081984  
1970,1539231,2465000,1471000,1471701,135443,708460  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1320789,378977,7322564  
2000,1537195,2485326,2229379,1332450,419728,8080879  
2010,1583873,2504705,2210722,1385108,451000,8175133  
2015,1444518,2636733,2339150,1459446,474558,8059405
```

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

```
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).

# Week 8: function parameters, github

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

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

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

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

# Week 8: function parameters, github

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

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

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

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

# Week 8: function parameters, github

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

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

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

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

# Week 8: function parameters, github

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

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

# 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

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)
```

```
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



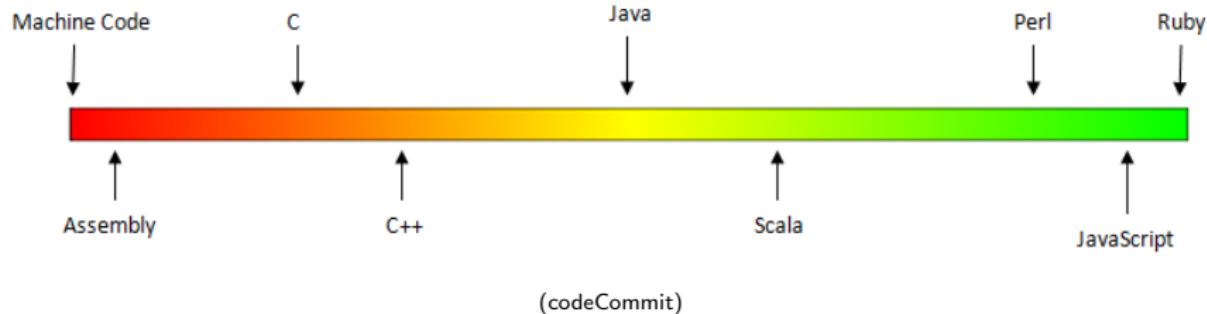
- Input/Output (I/O): `input()` and `print()`; pandas for CSV files
- Types:
  - ▶ Primitive: `int`, `float`, `bool`, `string`;
  - ▶ Container: lists (but not dictionaries/hashes or tuples)
- Objects: turtles (used but did not design our own)
- Loops: definite & indefinite
- Conditionals: `if-elif-else`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

# Today's Topics



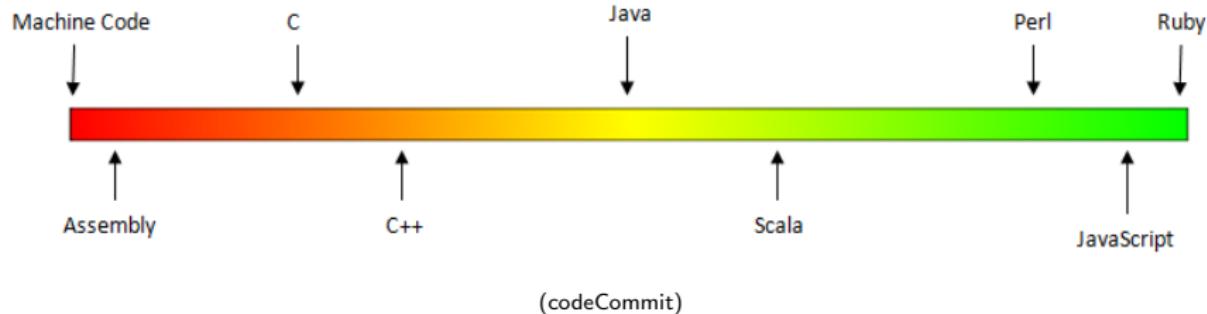
- Design Patterns: Searching
- Python Recap
- **Machine Language**
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

# Low-Level vs. High-Level Languages



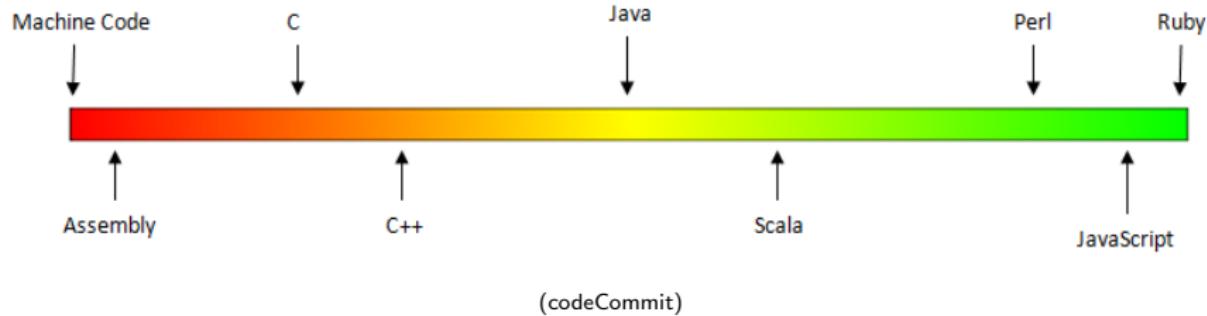
- Can view programming languages on a continuum.

# Low-Level vs. High-Level Languages



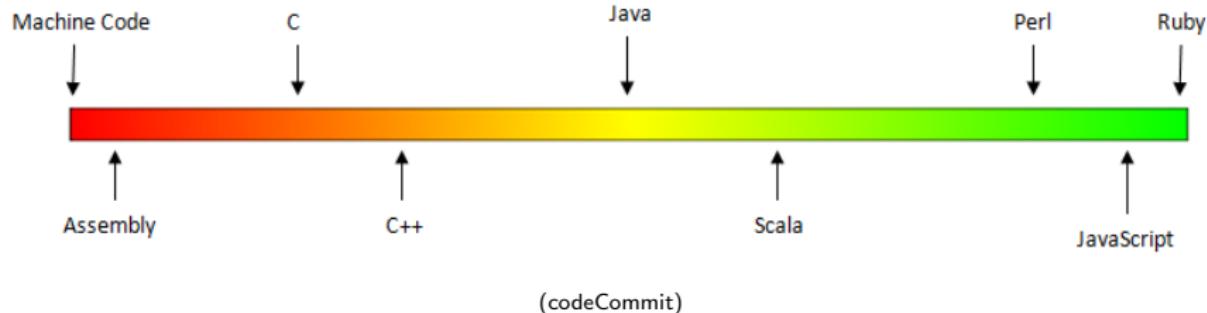
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

# Low-Level vs. High-Level Languages



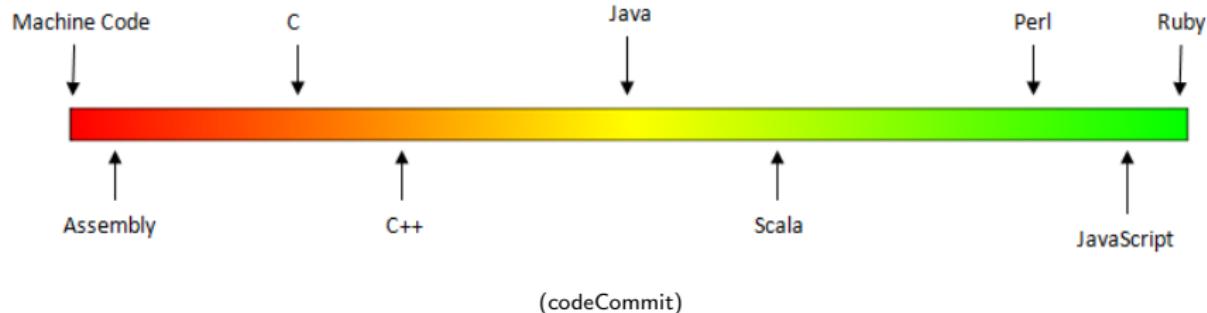
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

# Low-Level vs. High-Level Languages



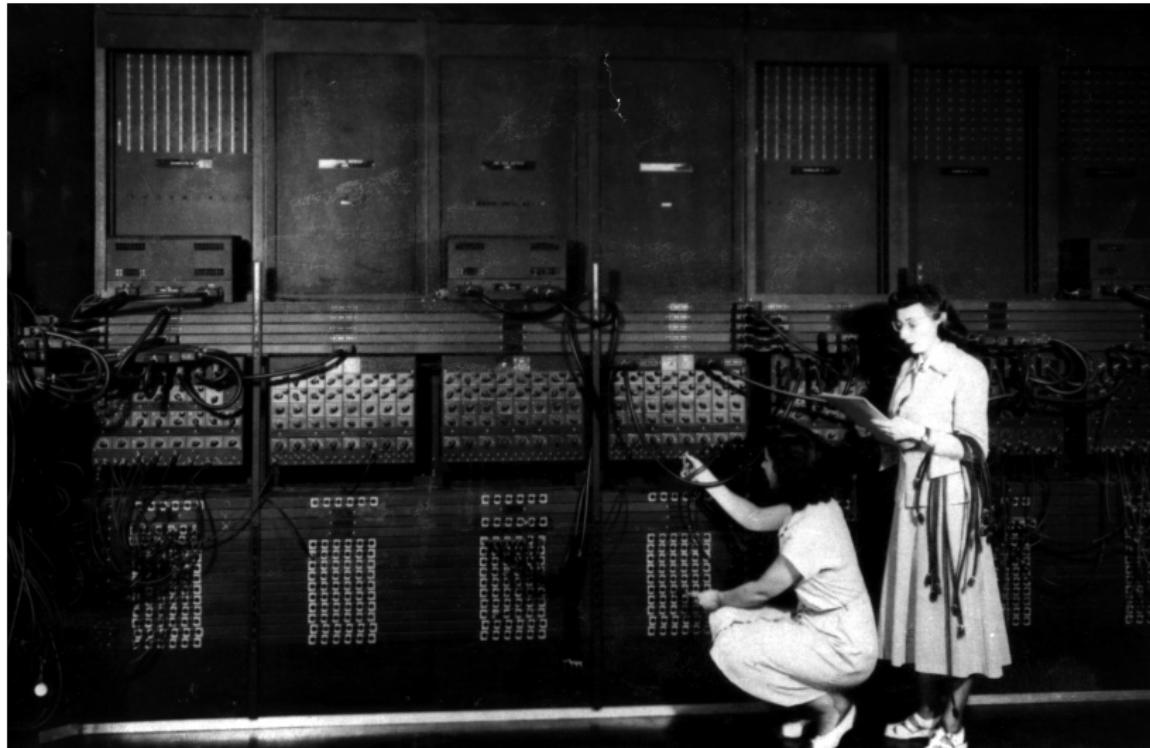
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
- Those that have strong abstraction (allow programming paradigms independent of the machine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

# Low-Level vs. High-Level Languages



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

# 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 00:UU .....
```

(wiki)

# Machine Language

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

The screenshot shows a terminal window with assembly code and its corresponding binary output. The assembly code is:

```
R 00200000 C2 30 [4:1]a 21-  
R 00200002 1B C0  
R 00200003 FB C0  
R 00200004 89 34 12 LDA #1234  
R 00200007 69 21 43 LDH #4321  
R 00200008 8F 03 7F 01 STA #17F03  
R 0020000E 8F 03 7F 03 CLD  
R 00200010 E2 30 SEP #38  
R 00200011 80 SWP  
R 20012  
T PB PC MUW#012C A .X .Y SP DP R8  
: 00 E012 00110000 0000 0000 0002 CFFF 0000 00  
$ 20000  
BREAK  
PB PC MUW#012C A .X .Y SP DP R8  
: 00 E012 00110000 5555 0000 0002 CFFF 0000 00  
$ 2003 27E3  
$007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
$007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

The binary output below the assembly code is:

```
PB PC MUW#012C A .X .Y SP DP R8  
: 00 E012 00110000 5555 0000 0002 CFFF 0000 00  
$ 2003 27E3  
$007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
$007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(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.



The screenshot shows a WeMIPS assembly editor interface. On the left, there is an assembly code editor with the following content:

```
    C2 30    REP $#30
    0020021B    CLD
    002003FB    SEI
    00200434 34 12    LDA #1234
    00200769 21 43    ADC #4321
    002008BF 03 7F 01    STA #017F03
    002009E8    CLD
    00200AE2 30    SEP $#30
    00201180    SWI
    A 2012
```

Below the assembly code is a memory dump window showing the memory state:

PC	Memory Address	Value
002000	00100000	0000 0000 0002 CFFF 0000 00
002001	00100001	0000 0000 0002 CFFF 0000 00
002002	00100002	0000 0000 0002 CFFF 0000 00
002003	00100003	0000 0000 0002 CFFF 0000 00
002004	00100004	0000 0000 0002 CFFF 0000 00
002005	00100005	0000 0000 0002 CFFF 0000 00
002006	00100006	0000 0000 0002 CFFF 0000 00
002007	00100007	0000 0000 0002 CFFF 0000 00
002008	00100008	0000 0000 0002 CFFF 0000 00
002009	00100009	0000 0000 0002 CFFF 0000 00
00200A	0010000A	0000 0000 0002 CFFF 0000 00
00200B	0010000B	0000 0000 0002 CFFF 0000 00
00200C	0010000C	0000 0000 0002 CFFF 0000 00
00200D	0010000D	0000 0000 0002 CFFF 0000 00
00200E	0010000E	0000 0000 0002 CFFF 0000 00
00200F	0010000F	0000 0000 0002 CFFF 0000 00
002010	00100010	0000 0000 0002 CFFF 0000 00
002011	00100011	0000 0000 0002 CFFF 0000 00
002012	00100012	0000 0000 0002 CFFF 0000 00

(wiki)

# Machine Language



The screenshot shows a computer screen with a window titled "WeMIPS". Inside the window, there is assembly code and a register dump.

**Assembly Code:**

```
R 00200000 C2 30 REP $#30
R 00200002 1B CLD
R 00200003 FB SEI
R 00200004 69 34 12 LDA #1234
R 00200007 69 21 43 LDH #4321
R 00200008 8F 03 7F 01 STA #17F03
R 0020000E 8F 03 7F 00 CLD
R 0020000F E2 30 SEP $#30
R 00200011 80 SWI
R 00200012
```

**Registers:**

Reg	PC	MAR	MDR	A	X	Y	SP	DP	RR
PB	00 0012	00110000	0000 0000	0002	CFFF	0000	00		
REG	00 2000								
BREAK									

**Memory Dump:**

Address	Value
00 000000	5555 0000 0002 CFFF 0000 00
00 000001	7FFF 2703 2703
00 000002	0000 0000 0000 0000 0000 0000 0000 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

Assembly code:

```
A 00200000 C2 38      REP    $#38
A 00200002 1B          CLD
A 00200003 FB          SEI
A 00200004 00 34 12    LDA    $#1234
A 00200007 69 21 43    LDH    $#4321
A 00200008 8F 03 7F 01  STW    $#017F03
A 0020000E 8F 03 7F 00  CLD
A 00200010 00 34 12    LDA    $#3412
A 00200011 00 34 12    LDA    $#3412
A 00200012
P0 PC     MM#000100C  A   X    Y    SP   DP   R0
: 00110000 00000000000000002 CFFF 0000 00
& 28000000
BREAK
P0 PC     MM#000100C  A   X    Y    SP   DP   R0
: 00110000 000000005555 00000 0002 CFFF 0000 00
& 28000000
>007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(wiki)

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

# "Hello World!" in Simplified Machine Language

Line: 3 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	<input checked="" type="checkbox"/> 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)





# MIPS Commands

The screenshot shows a MIPS assembly debugger interface. At the top, there are tabs for User, Show/Hide Demo, and several menu items like User Guide, Unit Tests, and Doc. Below the tabs are buttons for Addition, Counter, If-Else, Looper, Stack Test, and Hello World. Further down are buttons for Code Gen, Save String, Interactive, Binary/Decimal, Decimal/Binary, and Debug.

The main area contains assembly code:

```
1 # Shows "Hello world!" at the top of the stack
2 .text
3 .globl _start
4 .data
5 .word 1234, 5678, 1111 # a
6 .word 2222, 3333, 1000 # b
7 .word 5555, 6666, 1000 # c
8 .word 9999, 8888, 1000 # d
9 .word 4444, 7777, 1000 # e
10 .word 3333, 2222, 1000 # f
11 .word 6666, 1111, 1000 # g
12 .word 7777, 8888, 1000 # h
13 .word 9999, 5555, 1000 # i
14 .word 1111, 2222, 1000 # j
15 .word 3333, 4444, 1000 # k
16 .word 5555, 6666, 1000 # l
17 .word 7777, 8888, 1000 # m
18 .word 9999, 1111, 1000 # n
19 .word 2222, 3333, 1000 # o
20 .word 4444, 5555, 1000 # p
21 .word 6666, 7777, 1000 # q
22 .word 8888, 9999, 1000 # r
23 .word 1111, 2222, 1000 # s
24 .word 3333, 4444, 1000 # t
25 .word 5555, 6666, 1000 # u
26 .word 7777, 8888, 1000 # v
27 .word 9999, 1111, 0 # null
28 .word 1111, 2222, 1000 # w
29 .word 3333, 4444, 1000 # x
30 .word 5555, 6666, 1000 # y
31 .word 7777, 8888, 1000 # z
32 .word 9999, 5555, 0 # null
```

To the right of the code, there is a register table with columns S, T, A, V, Stack, and Log. The values for each register are:

S	T	A	V	Stack	Log
#0	10				
#1	9				
#2	8				
#3	22				
#4	66				
#5	81				
#6	807				
#7	418				

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

# MIPS Commands

The screenshot shows a software interface for MIPS assembly. At the top, there's a menu bar with 'File', 'Edit', 'Run', 'User Guide', 'Unit Tests', and 'Doc'. Below the menu are tabs for 'Addition', 'Calculator', 'Itax', 'Looper', 'Stack Test', 'Hello World', 'Code Gen', 'Save String', 'Interactive', 'Binary', 'Decimal', 'Hexadecimal', and 'Binary'. The 'Binary' tab is selected. A 'Debug' button is also present.

The main area contains assembly code:

```
1 # Shows "Hello world!" at the top of the stack
2 .text
3 .globl _start
4 .data
5 .word 1234,5678,1234,5678
6 .text
7 addi $t0,$zero,$111 # a
8 addi $t1,$zero,$111 # b
9 addi $t2,$zero,$111 # c
10 addi $t3,$zero,$111 # d
11 addi $t4,$zero,$111 # e
12 addi $t5,$zero,$111 # f
13 addi $t6,$zero,$111 # g
14 addi $t7,$zero,$111 # h
15 addi $t8,$zero,$111 # i
16 addi $t9,$zero,$111 # j
17 addi $t10,$zero,$111 # k
18 addi $t11,$zero,$111 # l
19 addi $t12,$zero,$111 # m
20 addi $t13,$zero,$111 # n
21 addi $t14,$zero,$111 # o
22 addi $t15,$zero,$111 # p
23 addi $t16,$zero,$111 # q
24 addi $t17,$zero,$111 # r
25 addi $t18,$zero,$111 # s
26 addi $t19,$zero,$111 # t
27 addi $t20,$zero,$111 # u
28 addi $t21,$zero,$111 # v
29 addi $t22,$zero,$111 # w
30 addi $t23,$zero,$111 # x
31 addi $t24,$zero,$111 # y
32 addi $t25,$zero,$111 # z
33 addi $t26,$zero,$111 # {main}
34 addi $t27,$zero,$111 # for point writing
35 addi $t28,$zero,$111 # print to the log
36 syscall
```

To the right of the assembly code is a table showing register values:

S	T	A	V	Stack	Log
\$0	10				
\$1	9				
\$2	8				
\$3	7				
\$4	6				
\$5	5				
\$6	407				
\$7	418				

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

# MIPS Commands

The screenshot shows the ShowFide Demo interface. At the top, there are tabs for User, ShowFide Demo, and other developer tools like Addition, Counter, Stack Test, Interactive, and so on. Below the tabs is a menu bar with File, Edit, Tools, Options, Help, and a Debug button. The main area contains assembly code for a "Hello World" program, starting with a stack setup and moving to a loop that prints "Hello world!". To the right of the code is a register dump table with columns for S, T, A, V, Stack, and Log. The registers shown are \$0 through \$7, with their values: \$0=10, \$1=9, \$2=8, \$3=22, \$4=66, \$5=67, \$6=807, and \$7=418.

```
1 # Shows "Hello world!" at the top of the stack
2 .data
3 .text
4 li $t0, 10
5 addi $t0, $zero, 100 # t
6 addi $t0, $zero, 100 # i
7 addi $t0, $zero, 100 # 1
8 addi $t0, $zero, 100 # l
9 addi $t0, $zero, 100 # o
10 addi $t0, $zero, 100 # w
11 addi $t0, $zero, 100 # r
12 addi $t0, $zero, 100 # l
13 addi $t0, $zero, 100 # d
14 addi $t0, $zero, 100 # ! (newline)
15 addi $t0, $zero, 100 # n
16 addi $t0, $zero, 100 # e
17 addi $t0, $zero, 100 # l
18 addi $t0, $zero, 100 # l
19 addi $t0, $zero, 100 # o
20 addi $t0, $zero, 100 # r
21 addi $t0, $zero, 100 # l
22 addi $t0, $zero, 100 # e
23 addi $t0, $zero, 100 # s
24 addi $t0, $zero, 100 # t
25 addi $t0, $zero, 100 # d
26 addi $t0, $zero, 100 # a
27 addi $t0, $zero, 100 # n
28 addi $t0, $zero, 100 # m
29 addi $t0, $zero, 100 # b
30 addi $t0, $zero, 100 # y
31 addi $t0, $zero, 100 # x
32 addi $t0, $zero, 100 # z
33 addi $t0, $zero, 100 # { (newline)
34 addi $t0, $zero, 100 # } (newline)
35 addi $t0, $zero, 100 # p (newline)
36 addi $t0, $zero, 100 # n (newline)
37 addi $t0, $zero, 100 # h (newline)
38 addi $t0, $zero, 100 # e (newline)
39 addi $t0, $zero, 100 # l (newline)
40 addi $t0, $zero, 100 # l (newline)
41 addi $t0, $zero, 100 # o (newline)
42 addi $t0, $zero, 100 # r (newline)
43 addi $t0, $zero, 100 # l (newline)
44 addi $t0, $zero, 100 # d (newline)
45 addi $t0, $zero, 100 # a (newline)
46 addi $t0, $zero, 100 # n (newline)
47 addi $t0, $zero, 100 # m (newline)
48 addi $t0, $zero, 100 # y (newline)
49 addi $t0, $zero, 100 # x (newline)
50 addi $t0, $zero, 100 # z (newline)
51 addi $t0, $zero, 100 # { (newline)
52 addi $t0, $zero, 100 # } (newline)
53 addi $t0, $zero, 100 # p (newline)
54 addi $t0, $zero, 100 # n (newline)
55 addi $t0, $zero, 100 # h (newline)
56 addi $t0, $zero, 100 # e (newline)
57 addi $t0, $zero, 100 # l (newline)
58 addi $t0, $zero, 100 # l (newline)
59 addi $t0, $zero, 100 # o (newline)
60 addi $t0, $zero, 100 # r (newline)
50 # print to the log
51 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:

# MIPS Commands

The screenshot shows a software interface for MIPS assembly simulation. At the top, there's a menu bar with 'File', 'Edit', 'Run', 'Help', 'User Guide', 'Unit Tests', and 'Doc'. Below the menu is a toolbar with buttons for 'Addition', 'Calculator', 'Itax', 'Looper', 'Stack Test', 'Hello World', 'Code Gen Save String', 'Interactive', 'Binary Decimal', 'Decimal Binary', and 'Debug'. A dropdown menu 'Step' is open, showing options like 'Run', 'Break', 'Stop', 'Next', 'Previous', 'Reset', 'Run to cursor', and 'Run to end'. The main area contains assembly code and a register dump table.

```
# Shows "Hello world" at the top of the stack
1    .text
2    .globl _start
3    .type _start, @function
4    _start:
5        addi   $t0, $zero, 124 # $t0 = 124
6        addi   $t1, $zero, 125 # $t1 = 125
7        addi   $t2, $zero, 126 # $t2 = 126
8        addi   $t3, $zero, 127 # $t3 = 127
9        addi   $t4, $zero, 128 # $t4 = 128
10       addi   $t5, $zero, 129 # $t5 = 129
11       addi   $t6, $zero, 130 # $t6 = 130
12       addi   $t7, $zero, 131 # $t7 = 131
13       addi   $t8, $zero, 132 # $t8 = 132
14       addi   $t9, $zero, 133 # $t9 = 133
15       addi   $t10, $zero, 134 # $t10 = 134
16       addi   $t11, $zero, 135 # $t11 = 135
17       addi   $t12, $zero, 136 # $t12 = 136
18       addi   $t13, $zero, 137 # $t13 = 137
19       addi   $t14, $zero, 138 # $t14 = 138
20       addi   $t15, $zero, 139 # $t15 = 139
21       addi   $t16, $zero, 140 # $t16 = 140
22       addi   $t17, $zero, 141 # $t17 = 141
23       addi   $t18, $zero, 142 # $t18 = 142
24       addi   $t19, $zero, 143 # $t19 = 143
25       addi   $t20, $zero, 144 # $t20 = 144
26       addi   $t21, $zero, 145 # $t21 = 145
27       addi   $t22, $zero, 146 # $t22 = 146
28       addi   $t23, $zero, 147 # $t23 = 147
29       addi   $t24, $zero, 148 # $t24 = 148
30       addi   $t25, $zero, 149 # $t25 = 149
31       addi   $t26, $zero, 150 # $t26 = 150
32       addi   $t27, $zero, 151 # $t27 = 151
# print to the log
33       syscall
```

S	T	A	V	Stack	Log
\$0				12	
\$1				9	
\$2				8	
\$3				22	
\$4				66	
\$5				67	
\$6				827	
\$7				418	

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

# MIPS Commands

The screenshot shows the ShowMe Demo interface. At the top, there are tabs for "ShowMe Demo", "User Guide", "Unit Tests", and "Doc". Below the tabs are several buttons: "Addition Counter", "Btav", "Looper", "Stack Test", "Hello World", "Code Gen Save String", "Interactive", "Binary Decimal", "Decimal Binary", and "Debug". The main area contains a text box with the following MIPS assembly code:

```
1 # Shows "Hello world" at the top of the stack
2
3 .data
4 .text
5 addi $t0,$zero,111 # 1
6 addi $t1,$zero,123 # 2
7 addi $t2,$zero,123 # 1
8 addi $t3,$zero,123 # 1
9 addi $t4,$zero,123 # 1
10 addi $t5,$zero,123 # 1
11 addi $t6,$zero,123 # 1
12 addi $t7,$zero,123 # 1
13 addi $t8,$zero,123 # 1
14 addi $t9,$zero,123 # 1
15 addi $t10,$zero,123 # 1
16 addi $t11,$zero,123 # 1
17 addi $t12,$zero,123 # 1
18 addi $t13,$zero,123 # 1
19 addi $t14,$zero,123 # 1
20 addi $t15,$zero,123 # 1
21 addi $t16,$zero,123 # 1
22 addi $t17,$zero,123 # 1
23 addi $t18,$zero,123 # 1
24 addi $t19,$zero,123 # 1
25 addi $t20,$zero,123 # 1
26 addi $t21,$zero,123 # 1
27 addi $t22,$zero,0 #(null)
28
29 addi $t23,$zero,4 # 4 is for print string
30 addi $t24,$zero,5 # print to the log
31
32 syscall
```

To the right of the assembly code is a register dump table with columns for S, T, A, V, Stack, and Log. The registers listed are \$0 through \$21, with their corresponding values.

S	T	A	V	Stack	Log
\$0	13				
\$1	9				
\$2	22				
\$3	66				
\$5	671				
\$6	807				
\$7	418				

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

# MIPS Commands



The screenshot shows a MIPS assembly debugger interface. The top menu bar includes 'File', 'Edit', 'Run', 'Help', 'Show/Hide Demo', 'User Guide', 'Unit Tests', and 'Doc'. Below the menu are tabs for 'Addition Counter', 'Btav', 'Looper', 'Stack Test', 'Hello World', 'Code Gen Save String', 'Interactive', 'Binary Decimal', 'Decimal Binary', and 'Debug'. The main window displays assembly code and a register dump.

**Assembly Code:**

```
# Shows "Hello world" at the top of the stack
1    .text
2    .globl _start
3    .type _start, @function
4    _start:
5        addi   $s0, $zero, 101 # $a
6        addi   $s1, $zero, 100 # $b
7        addi   $s2, $zero, 100 # $c
8        addi   $s3, $zero, 100 # $d
9        addi   $s4, $zero, 100 # $e
10       addi   $s5, $zero, 100 # $f
11       addi   $s6, $zero, 100 # $g
12       addi   $s7, $zero, 100 # $h
13       addi   $s8, $zero, 100 # $i
14       addi   $s9, $zero, 100 # $j
15       addi   $s10, $zero, 100 # $k
16       addi   $s11, $zero, 100 # $l
17       addi   $s12, $zero, 100 # $m
18       addi   $s13, $zero, 100 # $n
19       addi   $s14, $zero, 100 # $o
20       addi   $s15, $zero, 100 # $p
21       addi   $s16, $zero, 100 # $q
22       addi   $s17, $zero, 100 # $r
23       addi   $s18, $zero, 100 # $s
24       addi   $s19, $zero, 100 # $t
25       addi   $s20, $zero, 100 # $u
26       addi   $s21, $zero, 100 # $v
27       addi   $s22, $zero, 0 # ($all)
28       addi   $s23, $zero, 100 # $w
29       addi   $s24, $zero, 100 # $x
30       addi   $s25, $zero, 100 # $y
31       addi   $s26, $zero, 100 # $z
32       addi   $s27, $zero, 0 # print to the log
33       syscall
```

**Registers:**

S	T	A	V	Stack	Log
s0				101	
s1				9	
s2				8	
s3				20	
s4				100	
s5				100	
s6				100	
s7				100	
s8				100	
s9				100	
s10				100	
s11				100	
s12				100	
s13				100	
s14				100	
s15				100	
s16				100	
s17				100	
s18				100	
s19				100	
s20				100	
s21				100	
s22				100	
s23				100	
s24				100	
s25				100	
s26				100	
s27				100	

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

# MIPS Commands

The screenshot shows the ShowMe Demo interface. At the top, there are tabs for User, ShowMe Demo, Addition, Subtraction, Bitwise, Looper, Stack Test, Hello World, Code Gen, Save String, Interactive, Binary, Decimal, and Debug. Below the tabs is a text area containing MIPS assembly code. To the right is a table showing register values.

S	T	A	V	Stack	Log
\$0	10				
\$1	9				
\$2	8				
\$3	22				
\$4	60				
\$5	61				
\$6	807				
\$7	418				

Assembly code (from line 1 to 32):

```
1 # Shows "Hello world" at the top of the stack
2 .text
3 .globl _start
4 _start:
5 addi $t0, $zero, 100 # $t0 = 100
6 addi $t1, $zero, 100 # $t1 = 100
7 addi $t2, $zero, 100 # $t2 = 100
8 addi $t3, $zero, 100 # $t3 = 100
9 addi $t4, $zero, 100 # $t4 = 100
10 addi $t5, $zero, 100 # $t5 = 100
11 addi $t6, $zero, 100 # $t6 = 100
12 addi $t7, $zero, 100 # $t7 = 100
13 addi $t8, $zero, 100 # $t8 = 100
14 addi $t9, $zero, 100 # $t9 = 100
15 addi $t10, $zero, 100 # $t10 = 100
16 addi $t11, $zero, 100 # $t11 = 100
17 addi $t12, $zero, 100 # $t12 = 100
18 addi $t13, $zero, 100 # $t13 = 100
19 addi $t14, $zero, 100 # $t14 = 100
20 addi $t15, $zero, 100 # $t15 = 100
21 addi $t16, $zero, 100 # $t16 = 100
22 addi $t17, $zero, 100 # $t17 = 100
23 addi $t18, $zero, 100 # $t18 = 100
24 addi $t19, $zero, 100 # $t19 = 100
25 addi $t20, $zero, 100 # $t20 = 100
26 addi $t21, $zero, 100 # $t21 = 100
27 addi $t22, $zero, 100 # $t22 = 100
28 addi $t23, $zero, 100 # $t23 = 100
29 addi $t24, $zero, 100 # $t24 = 100
30 addi $t25, $zero, 100 # $t25 = 100
31 addi $t26, $zero, 100 # $t26 = 100
32 addi $t27, $zero, 100 # $t27 = 100
# print to the log
# 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.

# MIPS Commands

The screenshot shows the ShowMe Demo interface. At the top, there are tabs for User, ShowMe Demo, Addition, Subtraction, Bitwise, Looper, Stack Test, Hello World, Code Gen, Save String, Interactive, Binary, Decimal, and Debug. Below the tabs is a text area containing MIPS assembly code. To the right is a table showing register values.

S	T	A	V	Stack	Log
\$0	10				
\$1	9				
\$2	8				
\$3	22				
\$4	60				
\$5	61				
\$6	807				
\$7	418				

Assembly code:

```
1 # Shows "Hello world" at the top of the stack
2 .text
3 .globl _start
4 _start:
5    addi   $t0, $zero, 100 # 1
6    addi   $t1, $zero, 100 # 1
7    addi   $t2, $zero, 100 # 1
8    addi   $t3, $zero, 100 # 1
9    addi   $t4, $zero, 100 # 1
10   addi   $t5, $zero, 100 # 1
11   addi   $t6, $zero, 100 # 1
12   addi   $t7, $zero, 100 # 1
13   addi   $t8, $zero, 100 # 1
14   addi   $t9, $zero, 100 # 1
15   addi   $t10, $zero, 100 # 1
16   addi   $t11, $zero, 100 # 1
17   addi   $t12, $zero, 100 # 1
18   addi   $t13, $zero, 100 # 1
19   addi   $t14, $zero, 100 # 1
20   addi   $t15, $zero, 100 # 1
21   addi   $t16, $zero, 100 # 1
22   addi   $t17, $zero, 100 # 1
23   addi   $t18, $zero, 100 # 1
24   addi   $t19, $zero, 100 # 1
25   addi   $t20, $zero, 100 # 1
26   addi   $t21, $zero, 100 # 1
27   addi   $t22, $zero, 0 # (all)
28
29   addi   $t23, $zero, 4 # 4 is for print string
30   addi   $t24, $zero, 0 # print to the log
31
32   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

# MIPS Commands

The screenshot shows the ShowMe Demo interface. At the top, there are tabs for User, ShowMe Demo, Addition, Subtraction, Bitwise, Looper, Stack Test, Hello World, Code Gen, Save String, Interactive, Binary, Decimal, and Debug. Below the tabs is a text area containing MIPS assembly code. To the right is a table showing register values.

S	T	A	V	Stack	Log
\$0				10	
\$1				9	
\$2				8	
\$3				7	
\$4				6	
\$5				5	
\$6				4	
\$7				3	
\$8				2	
\$9				1	
\$10				0	
\$11				401	
\$12				410	

- **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)

# In Pairs or Triples:

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

The screenshot shows the WeMIPS debugger interface. At the top, there are tabs for 'Live' (selected), 'Dat', 'ShowHide Device' (dropdown menu), 'User Guide | Unit Tests | Docs', and navigation buttons. Below the tabs are buttons for 'Addition Doubler', 'Stax', 'Looper', 'Stack Test', and 'Hello World'. There are also links for 'Code Gen Save String', 'Interactive', 'Binary2 Decimal', 'Decimal2 Binary', and 'Debug'.

The main area displays assembly code:

```
# Store 'Hello world!' at the top of the stack
1    ADDI $t0,$zero,72 # H
2    ADDI $t1,$zero,101 # e
3    ADDI $t2,$zero,101 # n
4    ADDI $t3,$zero,101 # o
5    ADDI $t4,$zero,101 # r
6    ADDI $t5,$zero,101 # l
7    ADDI $t6,$zero,101 # w
8    ADDI $t7,$zero,101 # r
9    ADDI $t8,$zero,101 # l
10   ADDI $t9,$zero,101 # d
11   ADDI $t10,$zero,33 # !
12   ADDI $t11,$zero,0 # null
13   ADDI $t12,$zero,0 # null
14   ADDI $t13,$zero,0 # null
15   ADDI $t14,$zero,0 # null
16   ADDI $t15,$zero,0 # null
17   ADDI $t16,$zero,0 # null
18   ADDI $t17,$zero,0 # null
19   ADDI $t18,$zero,0 # null
20   ADDI $t19,$zero,0 # null
21   ADDI $t20,$zero,0 # null
22   ADDI $t21,$zero,0 # null
23   ADDI $t22,$zero,4 # 4 is for print string
24   ADDI $t23,$zero,0 # point to the log
25   syscall
```

Below the assembly code is a register table:

Step	Run	<input checked="" type="checkbox"/> Enable auto switching			
S	T	A	V	Stack	Log
s0	10				
s1	9				
s2	0				
s3	22				
s4	696				
s5	976				
s6	977				
s7	419				

The bottom of the interface has standard browser navigation controls.

(Demo with WeMIPS)

# Today's Topics



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

# Loops & Jumps in Machine Language

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



A screenshot of a debugger interface. On the left is the assembly code window, showing a series of machine instructions. On the right is the registers window, showing various CPU register values. The assembly code includes labels like `start:`, `loop:`, and `exit:`, along with various arithmetic and logical operations.

# Loops & Jumps in Machine Language

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



A screenshot of a hex editor application. The left pane shows a list of memory pages, each containing a series of memory addresses and their corresponding byte values. The right pane is a detailed view of a specific page, showing memory addresses from 0x00000000 to 0x0000000F. The bytes displayed are: 48 45 4C 4C 4D 4E 4F 4F 4A 4B 4C 4D 4E 4F 4F 4A 4B 4C. Below the address 0x00000000, there is a label 'Label' followed by a colon and the assembly instruction 'JMP Label'. The bottom of the window has standard file menu options like File, Edit, View, Insert, Options, and Help.

# 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.

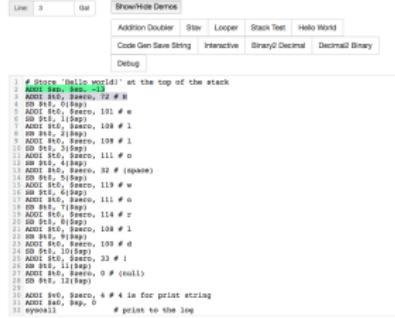


# Loops & Jumps in Machine Language

- 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



The screenshot shows the ShowHw Device interface. At the top, there are tabs for 'Live', '3', 'Dat', and 'ShowHw Device'. Below the tabs are buttons for 'Addition Doubler', 'Stax', 'Looper', 'Stack Test', and 'Hello World'. Further down are buttons for 'Code Gen Save String', 'Interactive', 'Binary2 Decimal', and 'Decimal2 Binary'. A 'Debug' button is also present. The main area contains assembly code and a debugger window.

```
# Store 'Hello world!' at the top of the stack
1    ADDI $t0, $zero, 72 # N
2    ADDI $t1, $zero, 101 # e
3    ADDI $t2, $zero, 101 # m
4    ADDI $t3, $zero, 101 # l
5    ADDI $t4, $zero, 101 # o
6    ADDI $t5, $zero, 101 # r
7    ADDI $t6, $zero, 101 # i
8    ADDI $t7, $zero, 101 # n
9    ADDI $t8, $zero, 101 # d
10   ADDI $t9, $zero, 33 # !
11   ADDI $t10, $zero, 0 # null
12   ADDI $t11, $zero, 4 # 4 is for print string
13   ADDI $t12, $zero, 0 # point to the log
14
15   # syscall
```

The debugger window shows the following registers:

Step	T	A	V	Stack	Log
S					
s0	10				
s1	9				
s2	8				
s3	22				
s4	695				
s5	976				
s6	977				
s7	419				

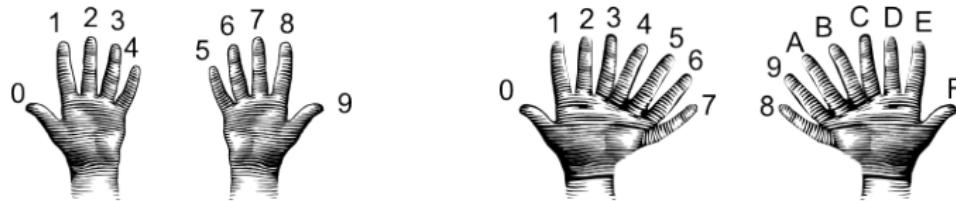
(Demo with WeMIPS)

# Today's Topics



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

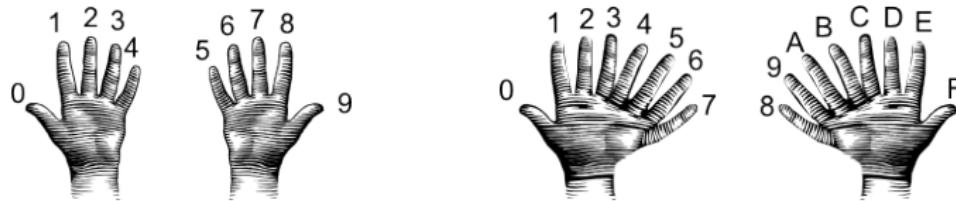
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.

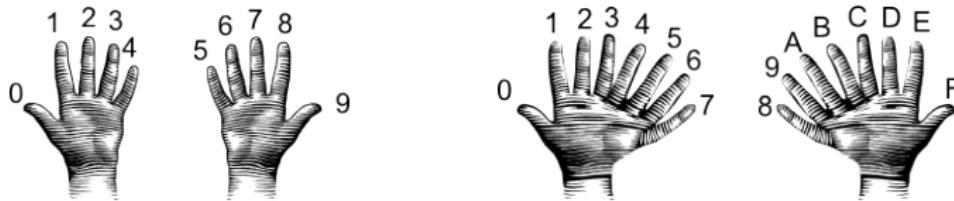
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.

# Hexadecimal to Decimal: Converting Between Bases

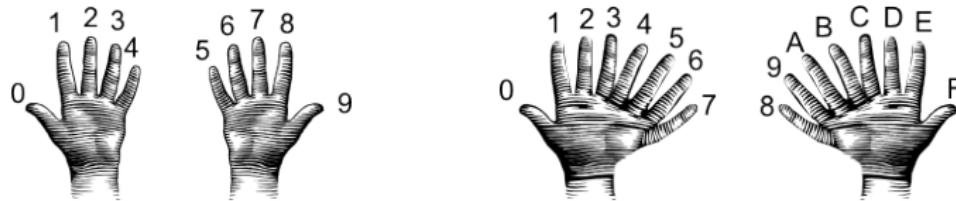


(from i-programmer.info)

- From hexadecimal to decimal:

- 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?

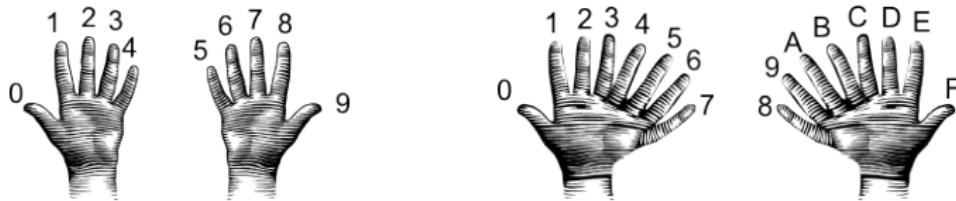
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:
  - 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.

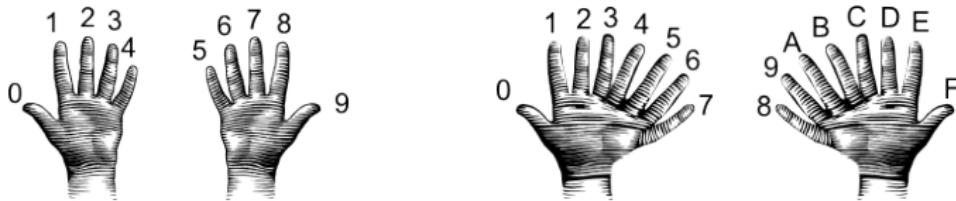
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:
  - 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.

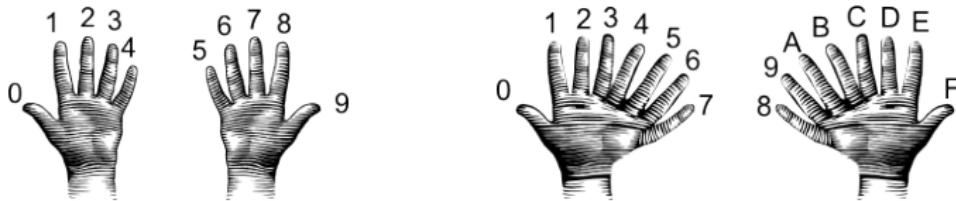
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:
  - 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:

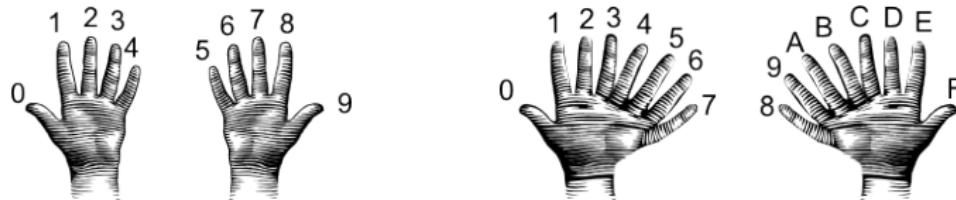
- Convert first digit to decimal and multiple by 16.
- Convert second digit to decimal and add to total.
- Example: what is 2A as a decimal number?

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:

- 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.

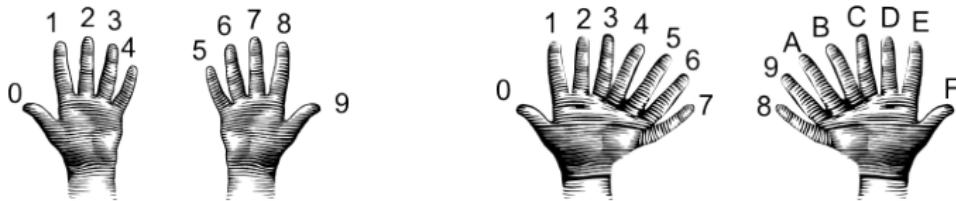
A in decimal digits is 10.

$32 + 10$  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:

- 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.

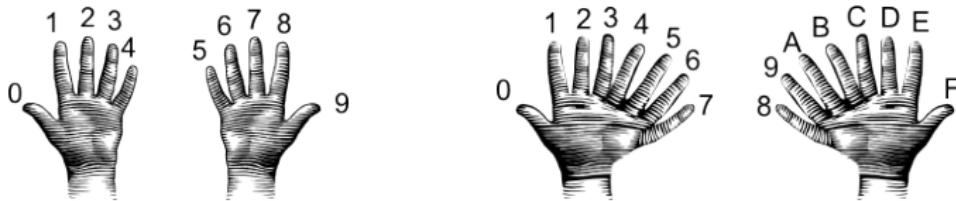
$32 + 10$  is 42.

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:

- 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.

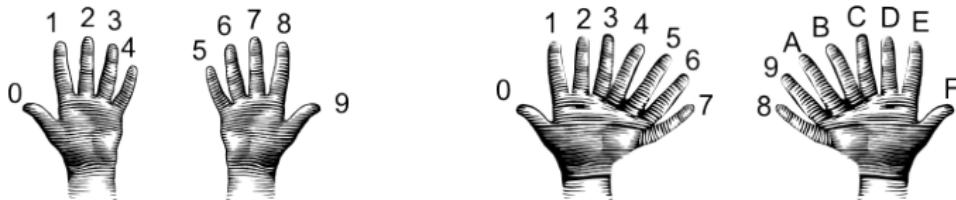
$32 + 10$  is 42.

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:

- 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.

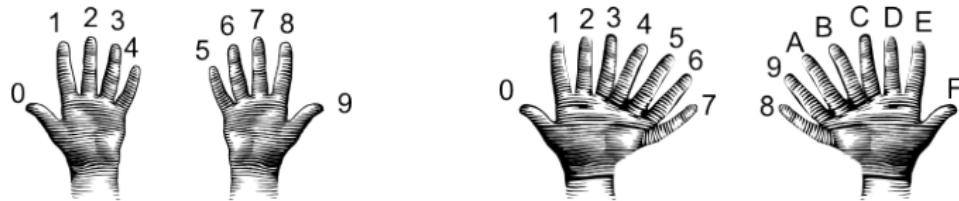
Answer is 42.

- 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:

- 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.

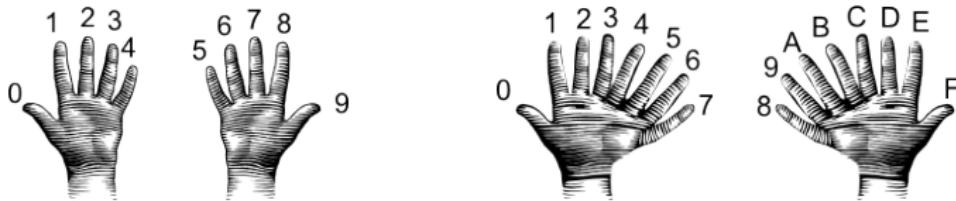
- Example: what is 99 as a decimal number?

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:

- 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?

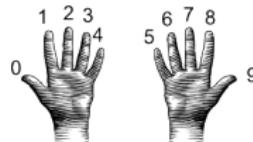
9 in decimal is 9.  $9 \times 16$  is 144.

9 in decimal digits is 9

$144 + 9$  is 153.

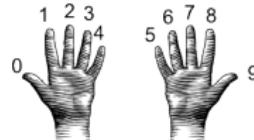
Answer is 153.

# Decimal to Binary: Converting Between Bases



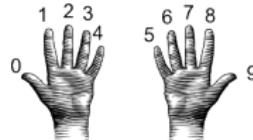
- From decimal to binary:
  - Divide by  $128 (= 2^7)$ . Quotient is the first digit.

# Decimal to Binary: Converting Between Bases



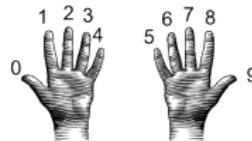
- From decimal to binary:
  - ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
  - ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.

# Decimal to Binary: Converting Between Bases



- 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.

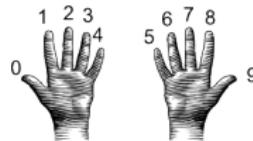
# Decimal to Binary: Converting Between Bases



- 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.

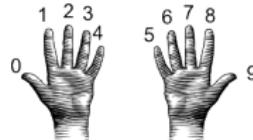
# Decimal to Binary: Converting Between Bases



- 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.

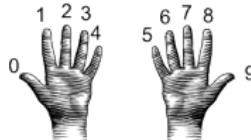
# Decimal to Binary: Converting Between Bases



- 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.

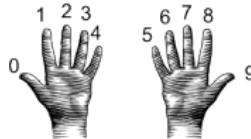
# Decimal to Binary: Converting Between Bases



- 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.

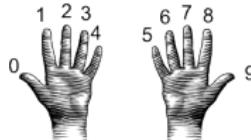
# Decimal to Binary: Converting Between Bases



- 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.

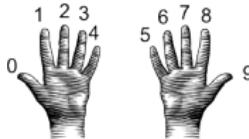
# Decimal to Binary: Converting Between Bases



- 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

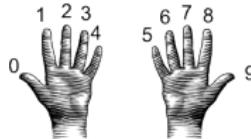


- From decimal to binary:

- 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

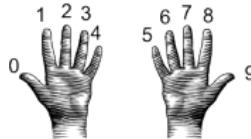


- From decimal to binary:

- ▶ 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



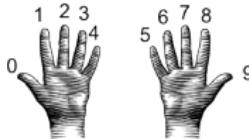
- From decimal to binary:

- ▶ 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



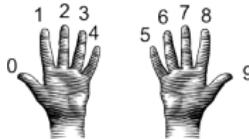
- From decimal to binary:

- ▶ 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



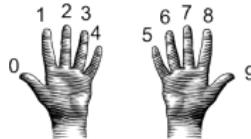
- From decimal to binary:

- 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



- From decimal to binary:

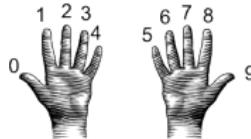
- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$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



- From decimal to binary:

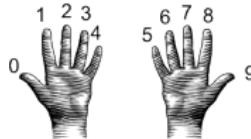
- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$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



- From decimal to binary:

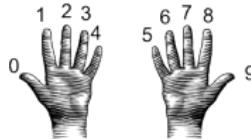
- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient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



- 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?

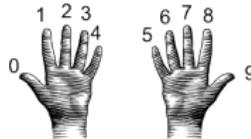
130/128 is 1 rem 2. First digit is 1: 1...

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



- 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?

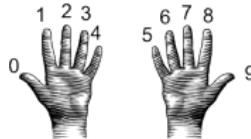
130/128 is 1 rem 2. First digit is 1: 1...

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



- 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?

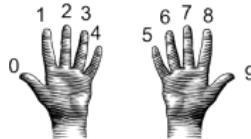
130/128 is 1 rem 2. First digit is 1: 1...

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



- 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...

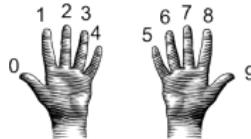
2/64 is 0 rem 2. Next digit is 0: 10...

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



- 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...

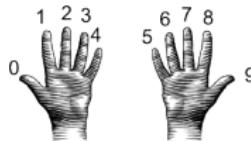
2/64 is 0 rem 2. Next digit is 0: 10...

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



- 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...

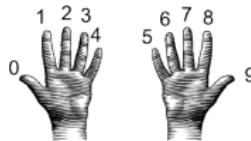
2/64 is 0 rem 2. Next digit is 0: 10...

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



- 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...

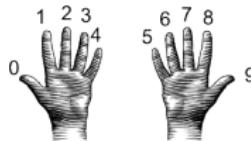
2/32 is 0 rem 2. Next digit is 0: 100...

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



- 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...

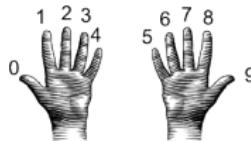
2/32 is 0 rem 2. Next digit is 0: 100...

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



- 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...

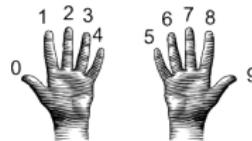
2/32 is 0 rem 2. Next digit is 0: 100...

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



- 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...

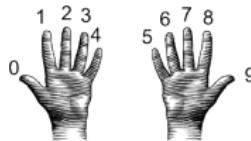
2/16 is 0 rem 2. Next digit is 0: 1000...

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



- 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...

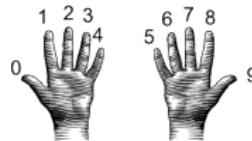
2/16 is 0 rem 2. Next digit is 0: 1000...

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



- 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...

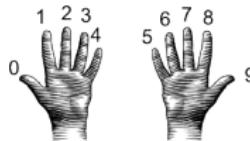
2/16 is 0 rem 2. Next digit is 0: 1000...

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



- 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...

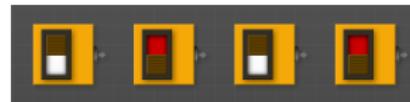
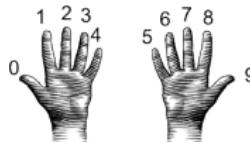
2/8 is 0 rem 2. Next digit is 0: 10000...

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



- 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...

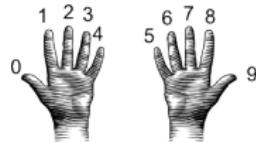
2/4 is 0 remainder 2. Next digit is 0: 100000...

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

Adding the last remainder:

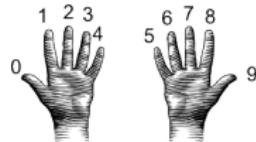
10000010

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

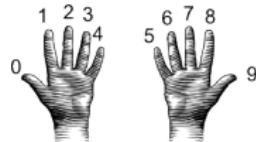
# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

$99/128 \text{ is } 0 \text{ rem } 99.$

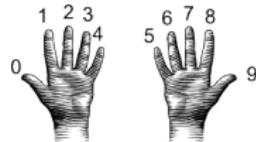
# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

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

# Decimal to Binary: Converting Between Bases

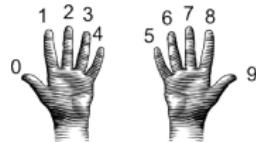


- Example: what is 99 in binary notation?

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

99/64 is 1 rem 35.

# Decimal to Binary: Converting Between Bases

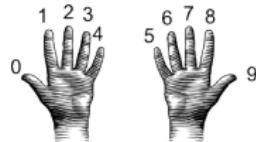


- Example: what is 99 in binary notation?

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

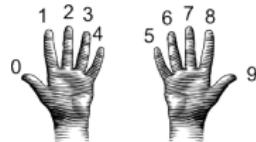


- Example: what is 99 in binary notation?

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



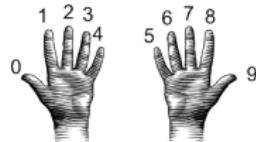
- Example: what is 99 in binary notation?

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



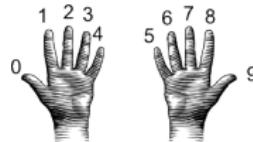
- Example: what is 99 in binary notation?

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



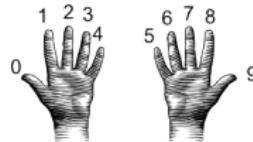
- Example: what is 99 in binary notation?

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: what is 99 in binary notation?

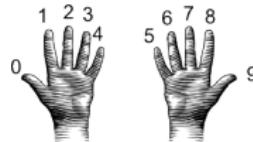
99/128 is 0 rem 99. First digit is 0: 0...

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: what is 99 in binary notation?

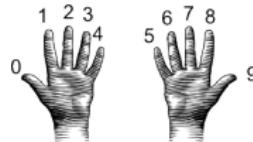
99/128 is 0 rem 99. First digit is 0: 0...

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: what is 99 in binary notation?

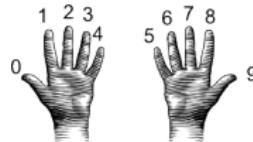
99/128 is 0 rem 99. First digit is 0: 0...

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: what is 99 in binary notation?

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

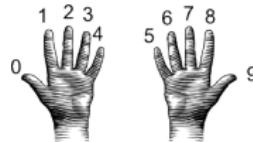
99/64 is 1 rem 35. Next digit is 1: 01...

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: what is 99 in binary notation?

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

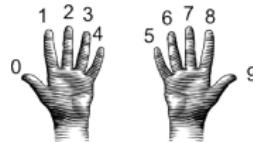
99/64 is 1 rem 35. Next digit is 1: 01...

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: what is 99 in binary notation?

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

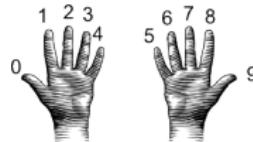
99/64 is 1 rem 35. Next digit is 1: 01...

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: 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...

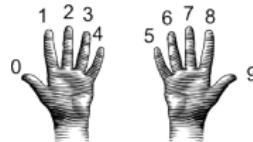
35/32 is 1 rem 3. Next digit is 1: 011...

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: 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...

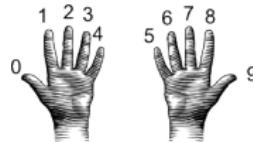
35/32 is 1 rem 3. Next digit is 1: 011...

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: 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...

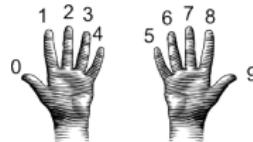
35/32 is 1 rem 3. Next digit is 1: 011...

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: 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...

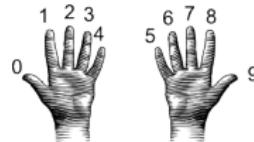
3/16 is 0 rem 3. Next digit is 0: 0110...

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: 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...

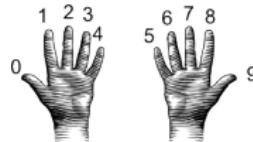
3/16 is 0 rem 3. Next digit is 0: 0110...

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: 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...

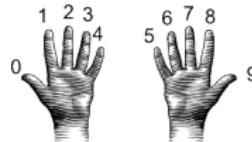
3/16 is 0 rem 3. Next digit is 0: 0110...

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: 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...

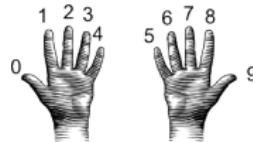
3/8 is 0 rem 3. Next digit is 0: 01100...

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: 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...

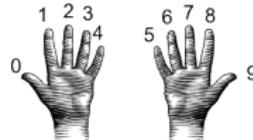
3/4 is 0 remainder 3. Next digit is 0: 011000...

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

Adding the last remainder: 01100011

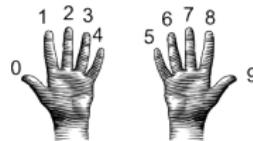
Answer is 1100011.

# Binary to Decimal: Converting Between Bases



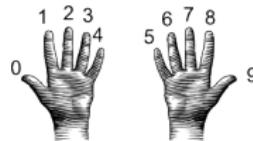
- From binary to decimal:
  - Set sum = last digit.

# Binary to Decimal: Converting Between Bases



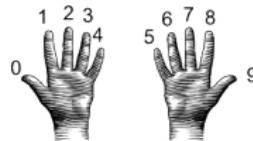
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by  $2^1$ . Add to sum.

# Binary to Decimal: Converting Between Bases



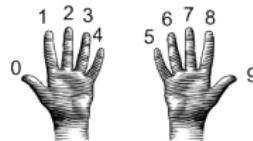
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by  $2 = 2^1$ . Add to sum.
  - Multiply next digit by  $4 = 2^2$ . Add to sum.

# Binary to Decimal: Converting Between Bases



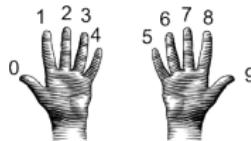
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by  $2^1$ . Add to sum.
  - Multiply next digit by  $2^2$ . Add to sum.
  - Multiply next digit by  $2^3$ . Add to sum.

# Binary to Decimal: Converting Between Bases



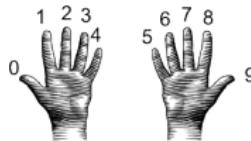
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by  $2^1$ . Add to sum.
  - Multiply next digit by  $4 = 2^2$ . Add to sum.
  - Multiply next digit by  $8 = 2^3$ . Add to sum.
  - Multiply next digit by  $16 = 2^4$ . Add to sum.

# Binary to Decimal: Converting Between Bases



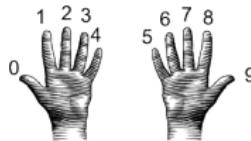
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by  $2^1$ . Add to sum.
  - Multiply next digit by  $4 = 2^2$ . Add to sum.
  - Multiply next digit by  $8 = 2^3$ . Add to sum.
  - Multiply next digit by  $16 = 2^4$ . Add to sum.
  - Multiply next digit by  $32 = 2^5$ . Add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

# Binary to Decimal: Converting Between Bases



- From binary to decimal:
  - ▶ Set sum = last digit.
  - ▶ Multiply next digit by  $2^1$ . 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.

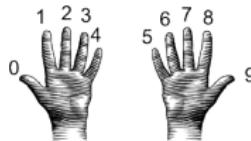
# Binary to Decimal: Converting Between Bases



- From binary to decimal:

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

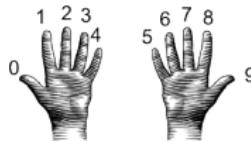
# Binary to Decimal: Converting Between Bases



- From binary to decimal:

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

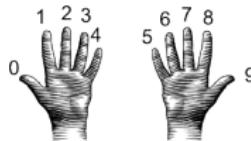


- From binary to decimal:

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

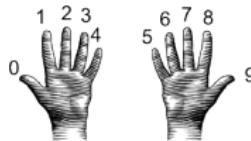


- From binary to decimal:

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

Sum starts with:                    1  
 $0 * 2 = 0$ . Add 0 to sum:

# Binary to Decimal: Converting Between Bases

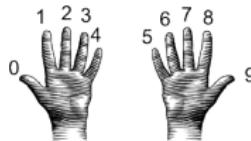


- From binary to decimal:

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



- From binary to decimal:

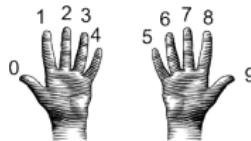
- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64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

Sum starts with:                    1

$0 * 2 = 0$ . Add 0 to sum:        1

$1 * 4 = 4$ . Add 4 to sum:

# Binary to Decimal: Converting Between Bases

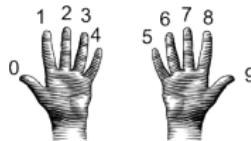


- From binary to decimal:

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

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

# Binary to Decimal: Converting Between Bases



- From binary to decimal:

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

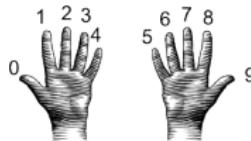
Sum starts with: 1

$0 * 2 = 0$ . Add 0 to sum: 1

$1 * 4 = 4$ . Add 4 to sum: 5

$1 * 8 = 8$ . Add 8 to sum:

# Binary to Decimal: Converting Between Bases

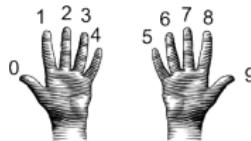


- From binary to decimal:

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

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

# Binary to Decimal: Converting Between Bases

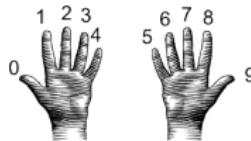


- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . 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  
 $0 \cdot 2 = 0$ . Add 0 to sum: 1  
 $1 \cdot 4 = 4$ . Add 4 to sum: 5  
 $1 \cdot 8 = 8$ . Add 8 to sum: 13  
 $1 \cdot 16 = 16$ . Add 16 to sum:

# Binary to Decimal: Converting Between Bases

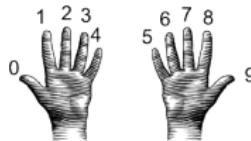


- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . 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  
 $0 \cdot 2 = 0$ . Add 0 to sum: 1  
 $1 \cdot 4 = 4$ . Add 4 to sum: 5  
 $1 \cdot 8 = 8$ . Add 8 to sum: 13  
 $1 \cdot 16 = 16$ . Add 16 to sum: 29

# Binary to Decimal: Converting Between Bases

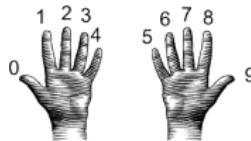


- From binary to decimal:

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

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

# Binary to Decimal: Converting Between Bases

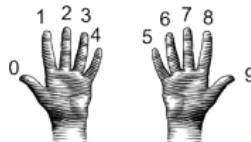


- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . 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
$0 \cdot 2 = 0$ . Add 0 to sum:	1
$1 \cdot 4 = 4$ . Add 4 to sum:	5
$1 \cdot 8 = 8$ . Add 8 to sum:	13
$1 \cdot 16 = 16$ . Add 16 to sum:	29
$1 \cdot 32 = 32$ . Add 32 to sum:	61

# Binary to Decimal: Converting Between Bases

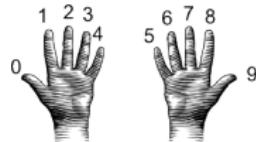


- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . 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
$0 \cdot 2 = 0$ . Add 0 to sum:	1
$1 \cdot 4 = 4$ . Add 4 to sum:	5
$1 \cdot 8 = 8$ . Add 8 to sum:	13
$1 \cdot 16 = 16$ . Add 16 to sum:	29
$1 \cdot 32 = 32$ . Add 32 to sum:	61

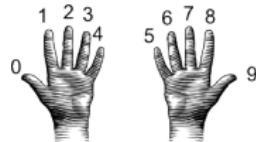
# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with:

# Binary to Decimal: Converting Between Bases

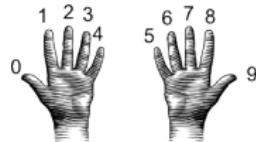


- Example: What is 10100100 in decimal?

Sum starts with: 0

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

# Binary to Decimal: Converting Between Bases

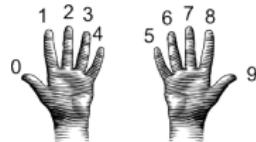


- Example: What is 10100100 in decimal?

Sum starts with: 0

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

# Binary to Decimal: Converting Between Bases



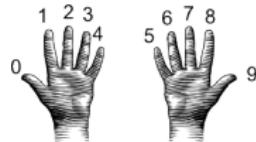
- Example: What is 10100100 in decimal?

Sum starts with: 0

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

$1 * 4 = 4$ . Add 4 to sum:

# Binary to Decimal: Converting Between Bases



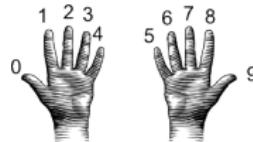
- Example: What is 10100100 in decimal?

Sum starts with: 0

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

$1 * 4 = 4$ . Add 4 to sum: 4

# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

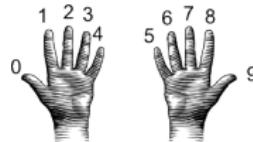
Sum starts with: 0

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

$1*4 = 4$ . Add 4 to sum: 4

$0*8 = 0$ . Add 0 to sum:

# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

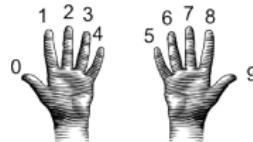
Sum starts with: 0

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

$1 * 4 = 4$ . Add 4 to sum: 4

$0 * 8 = 0$ . Add 0 to sum: 4

# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with: 0

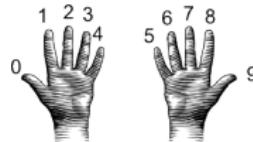
$0 \times 2 = 0$ . Add 0 to sum: 0

$1 \times 4 = 4$ . 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: What is 10100100 in decimal?

Sum starts with: 0

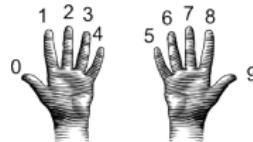
$0 \times 2 = 0$ . Add 0 to sum: 0

$1 \times 4 = 4$ . 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: What is 10100100 in decimal?

Sum starts with: 0

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

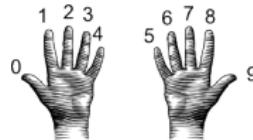
$1 \times 4 = 4$ . Add 4 to sum: 4

$0 \times 8 = 0$ . 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: What is 10100100 in decimal?

Sum starts with: 0

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

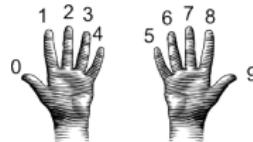
$1 \times 4 = 4$ . Add 4 to sum: 4

$0 \times 8 = 0$ . 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: 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

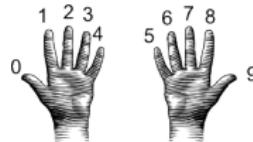
$0 \times 8 = 0$ . Add 0 to sum: 4

$0 \times 16 = 0$ . 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: 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

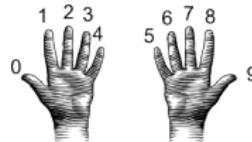
$0 \times 8 = 0$ . Add 0 to sum: 4

$0 \times 16 = 0$ . 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: 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

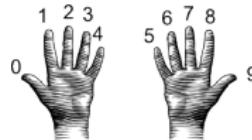
$0 \times 16 = 0$ . Add 0 to sum: 4

$1 \times 32 = 32$ . Add 32 to sum: 36

$0 \times 64 = 0$ . Add 0 to sum: 36

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

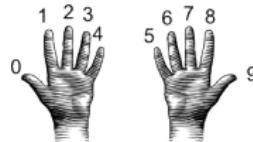
# Binary to Decimal: Converting Between Bases



- Example: 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

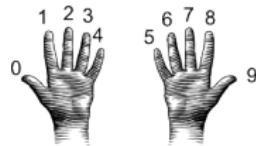


- Example: What is 10100100 in decimal?

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

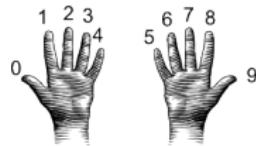
The answer is 164.

# Design Challenge: Incrementers



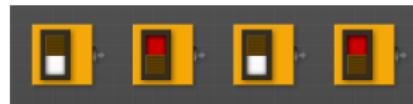
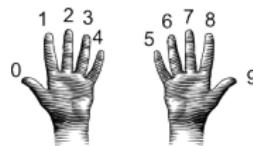
- Simplest arithmetic: add one ("increment") a variable.

# Design Challenge: Incrementers



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

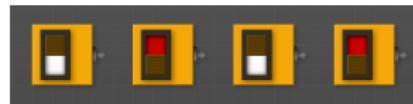
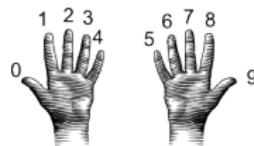
# Design Challenge: Incrementers



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

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

# Design Challenge: Incrementers

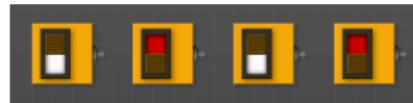
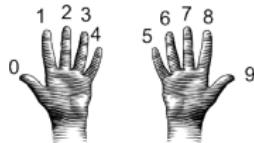


- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

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

- Challenge: Write an algorithm for incrementing numbers expressed as words.

# Design Challenge: Incrementers

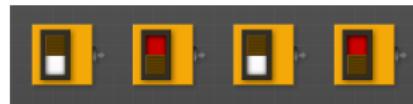
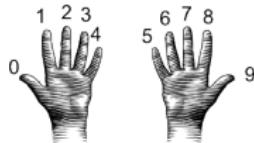


- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

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

- Challenge: Write an algorithm for incrementing numbers expressed as words.  
Example: "forty one" → "forty two"

# Design Challenge: Incrementers

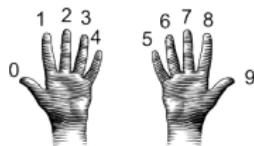


- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

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

- Challenge: Write an algorithm for incrementing numbers expressed as words.  
Example: "forty one" → "forty two"  
*Hint: Convert to numbers, increment, and convert back to strings.*

# Design Challenge: Incrementers

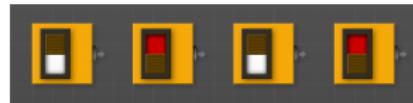
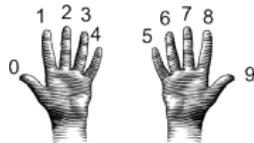


- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

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

- Challenge: Write an algorithm for incrementing numbers expressed as words.  
Example: "forty one" → "forty two"  
*Hint: Convert to numbers, increment, and convert back to strings.*
- Challenge: Write an algorithm for incrementing binary numbers.

# Design Challenge: Incrementers



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

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

- Challenge: Write an algorithm for incrementing numbers expressed as words.  
Example: "forty one" → "forty two"  
*Hint: Convert to numbers, increment, and convert back to strings.*
- Challenge: Write an algorithm for incrementing binary numbers.  
Example: "1001" → "1010"

# Recap

- On lecture slip, write down a topic you wish we had spent more time (and why).



# Recap



- On lecture slip, write down a topic you wish we had spent more time (and why).
- Searching through data is a common task— built-in functions and standard design patterns for this.

# Recap



- On lecture slip, write down a topic you wish we had spent more time (and why).
- 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



- On lecture slip, write down a topic you wish we had spent more time (and why).
- 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.
- Pass your lecture slips to the aisles for the UTAs to collect.

# Today's Topics



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

# 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.
  - ▶ 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 after spring break.

# 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 after spring break.
- 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

18 December 2018

### Exam Rules

- Show all your work. Your grade will be based on the work shown.
- You may bring one closed book and three notes with the exception of an 8.5" x 11" piece of paper filled with notes, programs, etc.
- When taking the exam, you may have white pens and pencils, and your note sheet.
- You may not use calculators, electronic calculators, mobile phones, or other electronic devices.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating on examinations, offering unfair advantages, and fabrication of records and official documents) as serious offenses against the integrity of the educational process. The Hunter College Code of Academic Integrity (Section 127.20) defines academic dishonesty and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedure.

<small>I understand that all cases of academic dishonesty will be reported to the Dean of Students and to the police authorities.</small>
Name _____
Signature _____
_____
_____
_____

# Exam Options

## Exam Times:

- Default: Regular Time: Monday, 16 December, 9-11am.
- Alternate Time: Reading Day, Friday, 13 December, 8:30am-10:30am.
- Accessibility Testing Center: Paperwork required. Must be completed on 13 December.

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

18 December 2018

### Exam Rules

- Show all your work. Your grade will be based on the work shown.
- You may bring one closed book and that book with the exception of an 8.5" x 11" piece of paper that with notes, programs, etc.
- When taking the exam, you may have whatever pens and pencils, and your note sheet.
- You may not use calculators, electronic calculators, online search, or other electronic devices.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating on examinations, offering unfair advantages, and fabrication of records and official documents) as serious offenses against the integrity of the educational process. The student handbook contains the Hunter College Academic Integrity Procedure.

<small>I understand that all cases of academic dishonesty will be reported to the Office of Student and Academic Conduct.</small>
Name _____
Signature _____
_____
_____
_____

# Exam Options

## Exam Times:

- Default: Regular Time: Monday, 16 December, 9-11am.
- Alternate Time: Reading Day, Friday, 13 December, 8:30am-10:30am.
- Accessibility Testing Center: Paperwork required. Must be completed on 13 December.

## Grading Options:

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

13 December 2018

### Exam Rules

- Show all your work. Your grade will be based on the work shown.
- You may bring one closed book and that book with the exception of an 8.5" x 11" piece of paper that with notes, programs, etc.
- When taking the exam, you may have whatever pens and pencils, and your note sheet.
- You may not use calculators, electronic calculators, online search, or other electronic devices.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating on examinations, offering unfair advantages, and fabrication of records and official documents) as serious offenses against the integrity of the educational process. The student handbook contains the policy on dealing with acts of academic dishonesty and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedure.

I understand that all cases of academic dishonesty will be reported to the Office of Student and Academic Conduct.	
Name:	
Signature:	
Date:	
Signature:	

# Exam Options

## Exam Times:

- Default: Regular Time: Monday, 16 December, 9-11am.
- Alternate Time: Reading Day, Friday, 13 December, 8:30am-10:30am.
- Accessibility Testing Center: Paperwork required. Must be completed on 13 December.

### Exam Rules

- Show all your work. Your grade will be based on the work shown.
- You may bring one closed book and that book 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 whatever pens and pencils, and your note sheet.
- You may not use calculators, electronic calculators, online search, or other electronic devices.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating on examinations, offering unfair advantages, and fabrication of records and official documents) as serious offenses against the integrity of the educational process. The student handbook contains the Hunter College Academic Integrity Procedure.

Students must affix their name to the line of handwriting and print below.
Name:
Signature:
Date:
Signature:

## Grading Options:

- Default: Letter Grade.
- Credit/NoCredit grade— availability depends on major and academic standing.

# Exam Options

## Exam Times:

- Default: Regular Time: Monday, 16 December, 9-11am.
- Alternate Time: Reading Day, Friday, 13 December, 8:30am-10:30am.
- Accessibility Testing Center: Paperwork required. Must be completed on 13 December.

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

13 December 2018

**Exam Rules**

- Show all your work. Your grade will be based on the work shown.
- You may bring one book and that book with the exception of an 8.5" x 11" piece of paper filled with notes, programs, etc.
- When taking the exam, you may have whatever pens and pencils, and your note sheet.
- You may not use calculators, electronic calculators, online search, or other electronic devices.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating on examinations, offering unfair advantages, and fabrication of records and official documents) as serious offenses against the integrity of the educational process. The student handbook contains the policy on Academic Honesty and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedure.

I understand that all cases of academic dishonesty will be reported to the Office of Student and Conduct Accountability.
Name _____
Signature _____
_____
_____

## Grading Options:

- Default: Letter Grade.
- Credit/NoCredit grade— availability depends on major and academic standing.

Forms for your choices (“pink slips”) available next lecture.

# Writing Boards



- Return writing boards as you leave...