

CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Today's Topics



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

Today's Topics



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

Predict what the code will do:

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

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

Python Tutor

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

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

(Demo with pythonTutor)

Design Pattern: Linear Search

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

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

- Example of **linear search**.

Design Pattern: Linear Search

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

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

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

Design Pattern: Linear Search

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

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

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

Design Pattern: Linear Search

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

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

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

Today's Topics



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

Python & Circuits Review: 10 Lectures in 10 Minutes



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

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

Lecture 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*

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

- Introduced comments & print():

```
#Name: Thomas Hunter
```

← These lines are comments

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

← (for us, not computer to read)

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

← (this one also)

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

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

- As well as definite loops & the turtle package:

The screenshot shows a Python IDE with a code editor on the left and a result window on the right. The code editor contains the following Python code:

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

The result window displays a purple hexagon with a turtle shape at each vertex, demonstrating the output of the code.

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

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

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

Lecture 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

Lecture 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

Lecture 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

Lecture 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

Lecture 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']

Lecture 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.

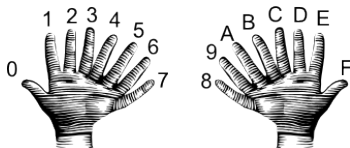
Lecture 2: variables, data types, more on loops & range()

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






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

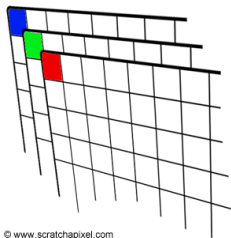
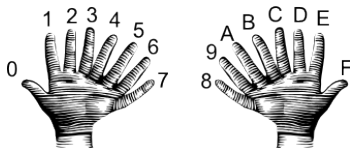
Lecture 3: colors, hex, slices, numpy & images

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








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

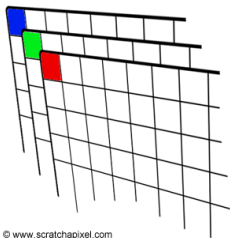
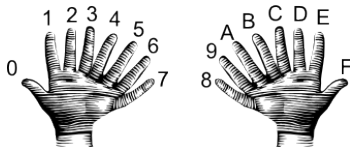
Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



© www.scratchapixel.com

Lecture 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

Lecture 4: design problem (cropping images) & decisions



Lecture 4: design problem (cropping images) & decisions



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

Lecture 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.

Lecture 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.

Lecture 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')
```

Lecture 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!")
```


Lecture 5: logical operators, truth tables & logical circuits

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

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

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



Lecture 6: structured data, pandas, & more design

```
Source: https://en.wikipedia.org/wiki/Demographics\_of\_New\_York\_City,....  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....  
.....  
.....  
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island>Total  
1698,4937,2017,,127,7881  
1771,21883,3623,,2847,28423  
1790,33131,4548,6159,1181,3827,49447  
1800,40515,5740,6642,1755,4543,79215  
1810,46373,6303,7444,2267,5347,119734  
1820,123706,11187,8246,2782,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312710,47613,34480,5344,10965,391114  
1850,515547,138882,18593,8032,15061,696115  
1860,813649,279122,32903,23593,25492,1174779  
1870,942282,419901,45648,37393,33029,1478183  
1880,1164673,599495,56559,51980,38991,1911698  
1890,1441216,838547,87050,88908,51693,2507414  
1900,1850093,1166582,152899,200507,67021,3437202  
1910,2331542,1634351,284041,430989,85969,4766883  
1920,2284183,2018356,449042,732018,116531,5620348  
1930,1867312,2560451,1079125,1565258,150346,6930446  
1940,1889924,2698285,1297634,1394711,174441,7454995  
1950,1960101,2738275,1550849,1452277,191555,7893257  
1960,1698281,2627319,1809578,1424815,221991,7781984  
1970,1539233,2602012,1996473,1471701,295443,7894862  
1980,1428285,2230936,1891325,1164872,352121,7071439  
1990,1487536,2300644,1951598,1203789,378977,7322564  
2000,1537195,2465326,2229379,1332450,443728,8008278  
2010,1648473,2504760,2230720,1385108,448730,8175133  
2015,1644518,2636738,2339155,1455444,474558,8550405
```

nycHistPop.csv

In Lab 6

Lecture 6: structured data, pandas, & more design

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

```
Source: https://en.wikipedia.org/wiki/Demographics\_of\_New\_York\_City,....
All population figures are consistent with present-day boundaries.....
First census after the consolidation of the five boroughs.....
.....
.....
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island>Total
1698,4937,2017,,127,7881
1771,21883,3623,,2847,28423
1790,35131,4548,6159,1181,3827,49447
1800,40515,5740,6642,1755,4543,79215
1810,46373,6303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,8048,3023,7082,242278
1840,312710,47613,34480,5344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32903,23593,25492,1174779
1870,942282,419801,45468,37393,33029,1478183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,115989,20507,67021,3437202
1910,2331542,1634351,284041,430989,85969,4766883
1920,2284183,2018356,449042,732018,116531,3620348
1930,1867312,2560401,1079129,1265258,150346,6950446
1940,1889924,2698285,1297634,1394711,174441,7454395
1950,1960101,2738275,1550849,1452277,191555,7893257
1960,1698281,2627319,1809578,1424815,221991,7781986
1970,1539233,2602012,1996473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332450,443728,8008278
2010,1648473,2504760,2230728,1385108,448738,8175133
2015,1644518,2636738,2339150,1455444,474558,8550405
```

nycHistPop.csv

In Lab 6

Lecture 6: structured data, pandas, & more design

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

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

```
Source: https://en.wikipedia.org/wiki/Demographics\_of\_New\_York\_City,....
All population figures are consistent with present-day boundaries.....
First census after the consolidation of the five boroughs.....
.....
.....
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island>Total
1698,4937,2017,,727,7681
1771,21863,3623,,2847,28423
1790,35131,4548,6159,1181,3827,49447
1800,40515,5740,6642,1755,4543,79215
1810,46373,6303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,8048,3023,7082,242278
1840,312710,47613,34480,5344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32903,23593,25492,1174779
1870,942282,419801,45648,37393,33829,1478183
1880,1164673,599495,56559,51980,38991,1911498
1890,1441216,838547,87050,88908,51493,2507414
1900,1650093,1166582,152899,20507,67021,3437202
1910,2331542,1634351,284041,430989,85969,4766883
1920,2284183,2018356,449042,732018,116531,5420348
1930,1867312,2580451,1079129,1565398,159346,6950446
1940,1889924,2698285,1297634,1394711,174441,7454395
1950,1940101,2738275,1550849,1452177,191555,7893257
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2402012,1986473,1471701,295443,7094862
1980,1428285,2230936,1891325,1148872,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1531795,2465326,2229379,1332450,443728,8008278
2010,1548473,2504760,2230728,1385108,448738,81751533
2015,1644518,2636738,2339150,1455444,474558,8550405
```

nycHistPop.csv

In Lab 6

Lecture 6: structured data, pandas, & more design

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

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

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

```
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,727,7881
1773,21883,3623,,2847,28423
1790,35131,4548,6159,1781,3827,49447
1800,40515,5740,6642,1755,4543,79215
1810,46373,6303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,8048,3023,7082,242278
1840,312710,47613,24480,5344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32903,23593,25492,1174779
1870,942282,419801,45648,37393,33029,1478183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1650093,1166582,152899,200507,67021,3437202
1910,2331542,1634351,284041,430989,85969,4766883
1920,2284183,2018356,448942,732016,116531,3420348
1930,1867312,2580451,1079129,1565398,159346,6506446
1940,1889924,2698285,1297634,1394711,174441,7454395
1950,1940101,2738275,1550849,1452277,191559,7893257
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2402012,1986473,1471701,295443,7094862
1980,1428285,2230936,1891325,1168972,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1531795,2465326,2229379,1332450,443728,8008278
2010,1548473,2504760,2230722,1385108,448730,8175133
2015,1644518,2636738,2339150,1455444,474558,8550405
```

nycHistPop.csv

In Lab 6

Lecture 6: structured data, pandas, & more design

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

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

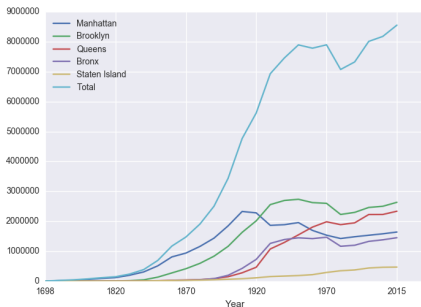
```
Source: https://en.wikipedia.org/wiki/Demographics\_of\_New\_York\_City,....
All population figures are consistent with present-day boundaries.....
First census after the consolidation of the five boroughs.....
.....
```

```
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,727,7481
1773,21883,3623,,2847,28423
1790,35131,4548,6159,1181,3827,49447
1800,40515,5740,6442,1755,4543,79215
1810,46373,6303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,8048,3023,7082,242278
1840,312710,47613,14480,5344,10965,393114
1850,515547,138882,18593,8032,15561,696115
1860,813649,279122,32903,23593,25492,1174779
1870,942282,419901,45468,37393,33529,1478103
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51493,2507414
1900,1650093,1146584,152899,20507,67021,3437202
1910,2331542,1634351,284041,430989,85949,4766883
1920,2284193,2018356,449042,732018,116531,5420348
1930,1867312,2560451,1079129,1565298,159346,6505446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738275,1550849,1452277,191559,7893257
1960,1698281,2627319,1809578,1424815,221991,7781986
1970,1539233,2402012,1996473,1471701,295443,7094862
1980,1428285,2230936,1891325,1148972,352121,7077439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332450,443728,8008278
2010,1648473,2504760,2230722,1385108,448730,8175133
2015,1644518,2636735,2339150,1455444,474558,8550405
```

nycHistPop.csv

In Lab 6

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



Lecture 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()
```

Lecture 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.

Lecture 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()`

Lecture 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:

Lecture 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")`

Lecture 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,

Lecture 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.

Lecture 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)
```

Lecture 8: function parameters, github

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

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

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

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

Lecture 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**.

Lecture 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**

Lecture 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.

Lecture 8: function parameters, github

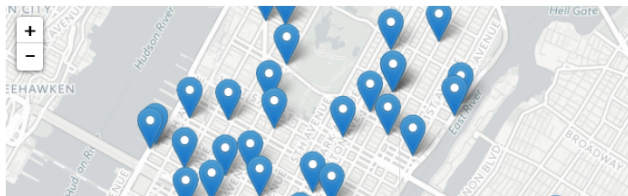
```
def totalWithTax(food,tip):  
    total = 0  
    tax = 0.0875  
    total = food + food * tax  
    total = total + tip  
    return(total)  
  
lunch = float(input('Enter lunch total: '))  
lTip = float(input('Enter lunch tip: '))  
lTotal = totalWithTax(lunch, lTip)  
print('Lunch total is', lTotal)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

Formal Parameters

Actual Parameters

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

Lecture 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)
```

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

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

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

```
import turtle
import random

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

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

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

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

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

```
import turtle
import random

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

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

Lecture 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.

Lecture 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.`

Lecture 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 Lectures in 10 Minutes



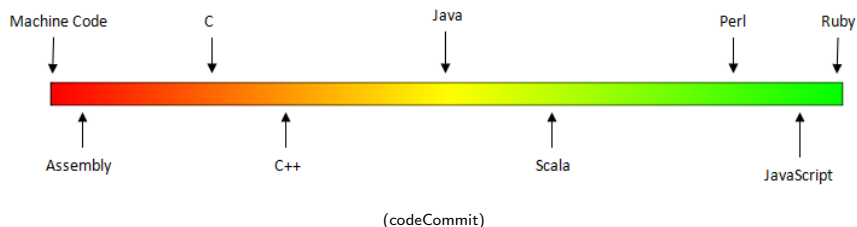
- Input/Output (I/O): `input()` and `print()`;
pandas for CSV files
- Types:
 - ▶ Primitive: `int`, `float`, `bool`, `string`;
 - ▶ Container: lists (but not dictionaries/hashtes or tuples)
- Objects: turtles (used but did not design our own)
- Loops: definite & indefinite
- Conditionals: `if-elif-else`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

Today's Topics



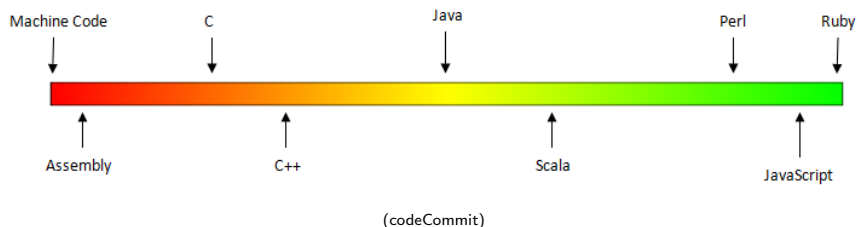
- Design Patterns: Searching
- Python Recap
- **Machine Language**
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

Low-Level vs. High-Level Languages



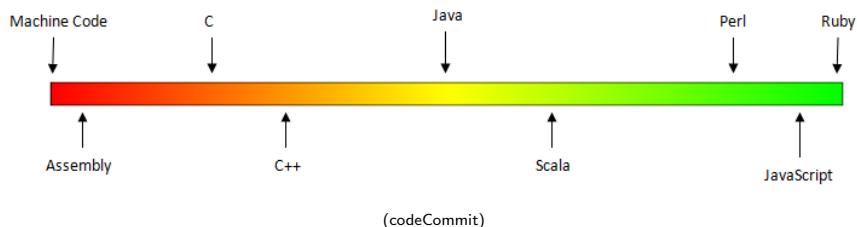
- Can view programming languages on a continuum.

Low-Level vs. High-Level Languages



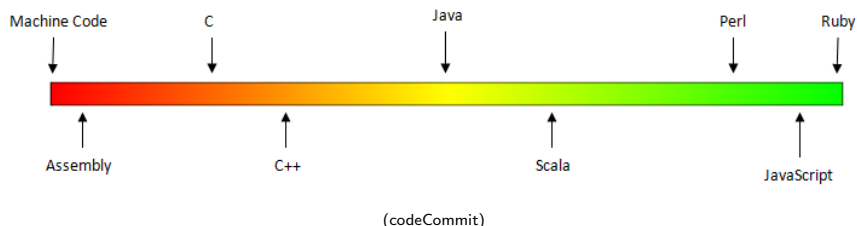
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

Low-Level vs. High-Level Languages



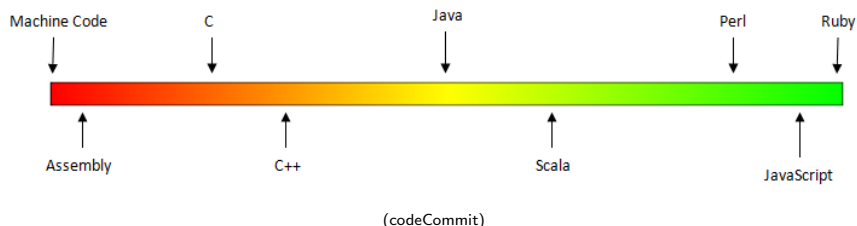
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

Low-Level vs. High-Level Languages



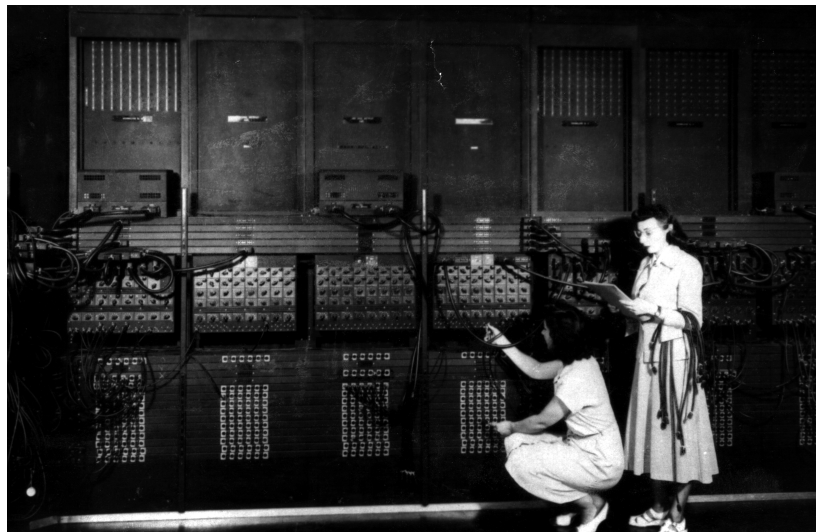
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
- Those that have strong abstraction (allow programming paradigms independent of the machine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

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

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

BREAK

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

(wiki)

Machine Language

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

```
002000 C2 30      REP #K30
R 002002 1B      CLC
R 002003 FB      SED
R 002004 09 34 12  LDR #01234
R 002007 09 21 43  RLC #04321
R 002009 0F 03 7F 01  STR #017FB3
R 00200C 00      CLD
R 00200F E2 30      SEP #K30
R 002011 0B      BRK
R 2012

P  PB PC Mem@PC  A   X   Y   SP   BP   BB
: 00 E012 00110000 0000 0000 0002 C7FF 0000 00
g 2008
BREAK
P  PB PC Mem@PC  A   X   Y   SP   BP   BB
: 00 2013 00110000 5555 0000 0002 C7FF 0000 00
n 1183 7FB3
007FB3 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(wiki)

Machine Language

```
002000 C2 30      REP #430
R 002002 1B      CLC
R 002003 FB      SED
R 002004 09 34 12  LSR #1234
R 002007 09 21 43  ROR #04321
R 002009 0F 03 7F 01  STB #017F03
R 00200C 00      CLD
R 00200F E2 30      SEP #430
R 002011 0B      BRK
R 2012

P0 PC Mem32C A X Y Z SP BP BB
: 00 2012 00110000 0000 0000 0002 C7FF 0000 00
$ 2000

BREAK

P0 PC Mem32C A X Y Z SP BP BB
: 00 2013 00110000 5555 0000 0002 C7FF 0000 00
n 1103 7F03
00FF03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(wiki)

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

Machine Language

```
002000 c2 30 REP #430  
R 002002 18 CLC  
R 002003 F8 SED  
R 002004 09 34 12 LSR #1234  
R 002007 09 21 43 RLC #4321  
R 002009 0F 03 7F 01 STW #017F03  
R 00200C 00 CLD  
R 00200F 02 30 SEW #430  
R 002011 00 BRK  
R 2012  
  
PB PC Mem32C A X Y SP BP BB  
: 00 2012 00110000 0000 0000 0000 0000 C7FF 0000 00  
0 200B  
BREAK  
  
PB PC Mem32C A X Y SP BP BB  
: 00 2013 00110000 5555 0000 0000 C7FF 0000 00  
0 1103 7F03  
00FF03 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.

Machine Language

```
002000 c2 30      REP #K30
002002 18          CLC
002003 F8          SED
002004 09 34 12  LSH #1234
002007 09 21 43  RSC #04321
002009 0F 03 7F 01  STA #017FB3
00200E 00          CLD
00200F 02 30      SEP #K30
002011 0B          BRK
02012

P  PC Mem32C  A   X   Y   SP  BP  B0
: 00 E012  00110000 0000 0000 0000 0000 0000 0000 00
$ 2000
BREAK
P  PC Mem32C  A   X   Y   SP  BP  B0
: 00 2013  00110000 5555 0000 0000 0000 0000 0000 00
n 1183 7183
00FF83 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(wiki)

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

“Hello World!” in Simplified Machine Language

Line: 3 Got

Show/Hide Demos

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

Addition Doubler Stav Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

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

Step Run Enable auto switching

S T A V Stack Log

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

(WeMIPS)

WeMIPS

Line 3 dis Show/Hide Demos User Guide | Unit Tests | Docs

Addition Doubler Star Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

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

Step Run Enable auto switching

S	T	A	V	Stack	Log
				a0:	10
				v0:	9
				a2:	9
				a3:	22
				a4:	905
				a5:	916
				a6:	927
				a7:	418

(Demo with WeMIPS)

MIPS Commands

The screenshot shows a MIPS simulator interface. On the left, there's a text area with assembly code. On the right, there's a register window titled 'Stop | Run | Create auto-switching' with columns for register names and their values.

```
1 # Show "hello world" at the top of the screen
2 ADDI $V0, $ZERO, 12 # R
3 SB $V0, $ZERO
4 ADDI $A0, $ZERO, 181 # R
5 SB $A0, $ZERO
6 ADDI $V0, $ZERO, 100 # L
7 SB $V0, $ZERO
8 ADDI $V0, $ZERO, 111 # R
9 SB $V0, $ZERO
10 ADDI $V0, $ZERO, 12 # (ASCII)
11 SB $V0, $ZERO
12 ADDI $V0, $ZERO, 119 # R
13 SB $V0, $ZERO
14 ADDI $V0, $ZERO, 111 # R
15 SB $V0, $ZERO
16 ADDI $V0, $ZERO, 114 # R
17 SB $V0, $ZERO
18 ADDI $V0, $ZERO, 109 # L
19 SB $V0, $ZERO
20 ADDI $V0, $ZERO, 100 # R
21 SB $V0, $ZERO
22 ADDI $V0, $ZERO, 10 # L
23 SB $V0, $ZERO
24 ADDI $V0, $ZERO, 0 # (null)
25 SB $V0, $ZERO
26 ADDI $V0, $ZERO, 4 # R in the print ending
27 ADDI $V0, $ZERO, 0
28 syscall
```

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

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

MIPS Commands

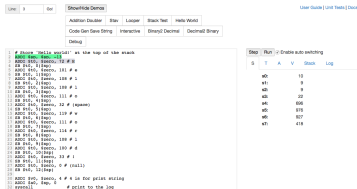
The screenshot shows a MIPS simulator interface. On the left, there is a text area containing assembly code. On the right, there is a register window titled 'Stop' with a table of registers.

```
1 # Show "hello world" as the top of the stack
2 ADDI $0, $zero, 12 # R
3 SB $0, 12($0)
4 ADDI $0, $zero, 181 # R
5 SB $0, 181($0)
6 ADDI $0, $zero, 100 # L
7 SB $0, 100($0)
8 ADDI $0, $zero, 100 # L
9 SB $0, 100($0)
10 ADDI $0, $zero, 111 # R
11 SB $0, 111($0)
12 ADDI $0, $zero, 12 # (space)
13 SB $0, 12($0)
14 ADDI $0, $zero, 100 # R
15 SB $0, 100($0)
16 ADDI $0, $zero, 111 # R
17 SB $0, 111($0)
18 ADDI $0, $zero, 114 # R
19 SB $0, 114($0)
20 ADDI $0, $zero, 100 # L
21 SB $0, 100($0)
22 ADDI $0, $zero, 100 # R
23 ADDI $0, $zero, 10 # L
24 SB $0, 10($0)
25 ADDI $0, $zero, 0 # (null)
26 SB $0, 0($0)
27 ADDI $0, $zero, 4 # 4 in the print string
28 ADDI $0, $0, 4 # print to the log
29 syscall
```

\$	T	A	V	Stack	Log
\$0				10	
\$1				8	
\$2				8	
\$3				22	
\$4				100	
\$5				100	
\$6				107	
\$7				418	

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

MIPS Commands



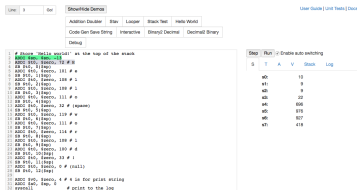
The screenshot shows a MIPS simulator interface. On the left, there is a text area containing assembly code. On the right, there is a register window titled 'Registers' with columns for 'S', 'T', 'A', 'V', 'Stack', and 'Log'. The register window shows values for registers \$0 through \$31.

```
1 # Show "Hello world" at the top of the stack
2 ADDI $0, $zero, 12 # R
3 SB $0, 0($0)
4 ADDI $0, $zero, 181 # R
5 SB $0, 1($0)
6 ADDI $0, $zero, 100 # L
7 SB $0, 2($0)
8 ADDI $0, $zero, 111 # R
9 SB $0, 3($0)
10 ADDI $0, $zero, 12 # (again)
11 SB $0, 4($0)
12 ADDI $0, $zero, 111 # R
13 SB $0, 5($0)
14 ADDI $0, $zero, 114 # R
15 SB $0, 6($0)
16 ADDI $0, $zero, 100 # L
17 SB $0, 7($0)
18 ADDI $0, $zero, 100 # R
19 SB $0, 8($0)
20 ADDI $0, $zero, 10 # L
21 SB $0, 9($0)
22 ADDI $0, $zero, 0 # (null)
23 SB $0, 10($0)
24 ADDI $0, $zero, 4 # 4 in the print string
25 ADDI $0, $zero, 4
26 syscall
# print to the log
```

S	T	A	V	Stack	Log
\$0				10	
\$1				8	
\$2				8	
\$3				22	
\$4				100	
\$5				110	
\$6				107	
\$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:

MIPS Commands



The screenshot shows a MIPS simulator interface. On the left, there is a text area containing assembly code. The code starts with a comment: `# Show 'hello world' at the top of the stack`. It then uses `ADDI $t0, $zero, 12` to set `$t0` to 12. A loop follows, using `ADDI $t1, $zero, 100` to set `$t1` to 100, and `ADDI $t2, $zero, 100` to set `$t2` to 100. The loop body consists of `ADDI $t3, $zero, 111`, `ADDI $t4, $zero, 12`, `ADDI $t5, $zero, 120`, `ADDI $t6, $zero, 111`, `ADDI $t7, $zero, 114`, `ADDI $t8, $zero, 100`, `ADDI $t9, $zero, 100`, `ADDI $t10, $zero, 100`, `ADDI $t11, $zero, 100`, `ADDI $t12, $zero, 100`, `ADDI $t13, $zero, 100`, `ADDI $t14, $zero, 100`, `ADDI $t15, $zero, 100`, `ADDI $t16, $zero, 100`, `ADDI $t17, $zero, 100`, `ADDI $t18, $zero, 100`, `ADDI $t19, $zero, 100`, `ADDI $t20, $zero, 100`, `ADDI $t21, $zero, 100`, `ADDI $t22, $zero, 100`, `ADDI $t23, $zero, 100`, `ADDI $t24, $zero, 100`, `ADDI $t25, $zero, 100`, `ADDI $t26, $zero, 100`, `ADDI $t27, $zero, 100`, `ADDI $t28, $zero, 100`, `ADDI $t29, $zero, 100`, `ADDI $t30, $zero, 100`, `ADDI $t31, $zero, 100`, `ADDI $t32, $zero, 100`, `ADDI $t33, $zero, 100`, `ADDI $t34, $zero, 100`, `ADDI $t35, $zero, 100`, `ADDI $t36, $zero, 100`, `ADDI $t37, $zero, 100`, `ADDI $t38, $zero, 100`, `ADDI $t39, $zero, 100`, `ADDI $t40, $zero, 100`, `ADDI $t41, $zero, 100`, `ADDI $t42, $zero, 100`, `ADDI $t43, $zero, 100`, `ADDI $t44, $zero, 100`, `ADDI $t45, $zero, 100`, `ADDI $t46, $zero, 100`, `ADDI $t47, $zero, 100`, `ADDI $t48, $zero, 100`, `ADDI $t49, $zero, 100`, `ADDI $t50, $zero, 100`, `ADDI $t51, $zero, 100`, `ADDI $t52, $zero, 100`, `ADDI $t53, $zero, 100`, `ADDI $t54, $zero, 100`, `ADDI $t55, $zero, 100`, `ADDI $t56, $zero, 100`, `ADDI $t57, $zero, 100`, `ADDI $t58, $zero, 100`, `ADDI $t59, $zero, 100`, `ADDI $t60, $zero, 100`, `ADDI $t61, $zero, 100`, `ADDI $t62, $zero, 100`, `ADDI $t63, $zero, 100`, `ADDI $t64, $zero, 100`, `ADDI $t65, $zero, 100`, `ADDI $t66, $zero, 100`, `ADDI $t67, $zero, 100`, `ADDI $t68, $zero, 100`, `ADDI $t69, $zero, 100`, `ADDI $t70, $zero, 100`, `ADDI $t71, $zero, 100`, `ADDI $t72, $zero, 100`, `ADDI $t73, $zero, 100`, `ADDI $t74, $zero, 100`, `ADDI $t75, $zero, 100`, `ADDI $t76, $zero, 100`, `ADDI $t77, $zero, 100`, `ADDI $t78, $zero, 100`, `ADDI $t79, $zero, 100`, `ADDI $t80, $zero, 100`, `ADDI $t81, $zero, 100`, `ADDI $t82, $zero, 100`, `ADDI $t83, $zero, 100`, `ADDI $t84, $zero, 100`, `ADDI $t85, $zero, 100`, `ADDI $t86, $zero, 100`, `ADDI $t87, $zero, 100`, `ADDI $t88, $zero, 100`, `ADDI $t89, $zero, 100`, `ADDI $t90, $zero, 100`, `ADDI $t91, $zero, 100`, `ADDI $t92, $zero, 100`, `ADDI $t93, $zero, 100`, `ADDI $t94, $zero, 100`, `ADDI $t95, $zero, 100`, `ADDI $t96, $zero, 100`, `ADDI $t97, $zero, 100`, `ADDI $t98, $zero, 100`, `ADDI $t99, $zero, 100`, `ADDI $t100, $zero, 100`. The code ends with `ADDI $t0, $zero, 4` (a note says "in the print window") and `ADDI $t0, $zero, 4` (a note says "print to the log").

On the right, there is a register window titled "Registers" with a table showing the values of registers \$0 through \$31. The values are: \$0: 0, \$1: 0, \$2: 0, \$3: 0, \$4: 0, \$5: 0, \$6: 0, \$7: 0, \$8: 0, \$9: 0, \$10: 0, \$11: 0, \$12: 0, \$13: 0, \$14: 0, \$15: 0, \$16: 0, \$17: 0, \$18: 0, \$19: 0, \$20: 0, \$21: 0, \$22: 0, \$23: 0, \$24: 0, \$25: 0, \$26: 0, \$27: 0, \$28: 0, \$29: 0, \$30: 0, \$31: 0.

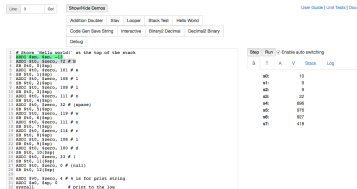
- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': `$s0`, `$s1`, `$t0`, `$t1`,...
- **R Instructions:** Commands that use data in the registers:
`add $s1, $s2, $s3`

MIPS Commands

The screenshot shows a MIPS simulator interface. On the left, there is a text area containing assembly code. The code starts with a comment: `# Show 'hello world' at the top of the stack`. It then uses `ADDI $t0, $zero, 12` to set register `$t0` to 12. Subsequent instructions use `ADDI $s0, $zero, 181` through `ADDI $s1, $zero, 114` to load values into registers `$s0` through `$s1`. The code concludes with `ADDI $t0, $zero, 4` and `syscall`. On the right, a register window titled "Registers" shows the current state of registers `$t0` through `$t7`. The values are: `$t0: 12`, `$t1: 0`, `$t2: 0`, `$t3: 0`, `$t4: 0`, `$t5: 0`, `$t6: 0`, and `$t7: 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 simulator interface. On the left, there is a text area containing assembly code. On the right, there is a register window showing the state of registers \$0 through \$31.

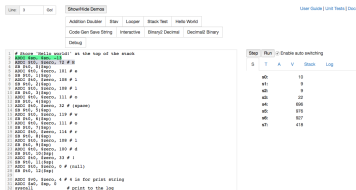
```
1 # Show "hello world" as the top of the stack
2 ADDI $0, $zero, 12 # R
3 SB $0, 0($zero), 12 # S
4 ADDI $0, $zero, 10 # R
5 SB $0, 4($zero), 10 # S
6 ADDI $0, $zero, 100 # R
7 SB $0, 8($zero), 100 # S
8 ADDI $0, $zero, 111 # R
9 SB $0, 12($zero), 111 # S
10 ADDI $0, $zero, 12 # (same)
11 SB $0, 16($zero), 12 # S
12 ADDI $0, $zero, 111 # R
13 SB $0, 20($zero), 111 # S
14 ADDI $0, $zero, 114 # R
15 SB $0, 24($zero), 114 # S
16 ADDI $0, $zero, 100 # R
17 SB $0, 28($zero), 100 # S
18 ADDI $0, $zero, 10 # R
19 SB $0, 32($zero), 10 # S
20 ADDI $0, $zero, 0 # (null)
21 SB $0, 36($zero), 0 # (null)
22 ADDI $0, $zero, 4 # 4 in the print string
23 ADDI $0, $0, 4
24 syscall
```

The register window on the right shows the following values:

Register	Value
\$0	12
\$1	0
\$2	0
\$3	100
\$4	111
\$5	12
\$6	111
\$7	114
\$8	100
\$9	10
\$10	0
\$11	4

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

MIPS Commands



The screenshot shows a MIPS simulator window with a menu bar (File, Edit, Show/Hide/Details, Add/Run/Debugger, Run, Loader, Stack View, Help/World) and a toolbar (Code Gen, Save String, Interactive, Disasm/Decimal, Decimal/Binary, Debug). The main area displays assembly code with comments:

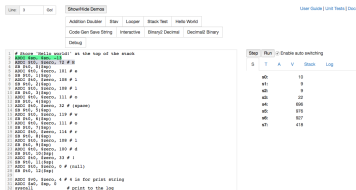
```
1 # Show "hello world" at the top of the stack
2 ADDI $s0, $zero, 12 # R
3 SB $s0, 0($zero)
4 ADDI $s0, $zero, 181 # R
5 SB $s0, 1($zero)
6 ADDI $s0, $zero, 100 # R
7 SB $s0, 2($zero)
8 ADDI $s0, $zero, 100 # L
9 SB $s0, 3($zero)
10 ADDI $s0, $zero, 111 # R
11 SB $s0, 4($zero)
12 ADDI $s0, $zero, 12 # (same)
13 SB $s0, 5($zero)
14 ADDI $s0, $zero, 111 # R
15 SB $s0, 6($zero)
16 ADDI $s0, $zero, 114 # R
17 SB $s0, 7($zero)
18 ADDI $s0, $zero, 100 # L
19 SB $s0, 8($zero)
20 ADDI $s0, $zero, 100 # L
21 SB $s0, 9($zero)
22 ADDI $s0, $zero, 10 # L
23 SB $s0, 10($zero)
24 ADDI $s0, $zero, 0 # (null)
25 SB $s0, 11($zero)
26 ADDI $s0, $zero, 4 # 4 in the print string
27 ADDI $s0, $zero, 6
28 syscall
# print to the top
```

On the right, a register window shows the state of registers \$0 through \$31:

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

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

MIPS Commands



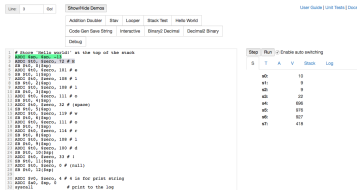
The screenshot shows a MIPS simulator window with a menu bar (File, Edit, Show/Hide/Details, Add/Run, Register, Stack View, Help/Watch) and a toolbar. The main area displays assembly code with comments. A register window on the right shows the state of registers \$0 through \$31.

```
1 # Show "Hello world" at the top of the stack
2 ADDI $0, $zero, 12 # R
3 SB $0, 0($zero)
4 ADDI $0, $zero, 181 # R
5 SB $0, 1($zero)
6 ADDI $0, $zero, 100 # R
7 SB $0, 2($zero)
8 ADDI $0, $zero, 100 # L
9 SB $0, 3($zero)
10 ADDI $0, $zero, 111 # R
11 SB $0, 4($zero)
12 ADDI $0, $zero, 12 # (wrong)
13 SB $0, 5($zero)
14 ADDI $0, $zero, 111 # R
15 SB $0, 6($zero)
16 ADDI $0, $zero, 114 # R
17 SB $0, 7($zero)
18 ADDI $0, $zero, 100 # L
19 SB $0, 8($zero)
20 ADDI $0, $zero, 100 # L
21 SB $0, 9($zero)
22 ADDI $0, $zero, 10 # L
23 SB $0, 10($zero)
24 ADDI $0, $zero, 0 # (null)
25 SB $0, 11($zero)
26 ADDI $0, $zero, 4 # in the print string
27 ADDI $0, $zero, 6
28 syscall
# print to the log
```

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

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

MIPS Commands



The screenshot shows a MIPS simulator interface. On the left, there is a text area containing assembly code. The code includes comments and instructions such as `addi $s0, $zero, 12`, `add $t0, $s0, 100`, `addi $s1, $s0, 100`, `addi $s2, $s0, 100`, `addi $s3, $s0, 100`, `addi $s4, $s0, 100`, `addi $s5, $s0, 100`, `addi $s6, $s0, 100`, `addi $s7, $s0, 100`, `addi $s8, $s0, 100`, `addi $s9, $s0, 100`, `addi $s10, $s0, 100`, `addi $s11, $s0, 100`, `addi $s12, $s0, 100`, `addi $s13, $s0, 100`, `addi $s14, $s0, 100`, `addi $s15, $s0, 100`, `addi $s16, $s0, 100`, `addi $s17, $s0, 100`, `addi $s18, $s0, 100`, `addi $s19, $s0, 100`, `addi $s20, $s0, 100`, `addi $s21, $s0, 100`, `addi $s22, $s0, 100`, `addi $s23, $s0, 100`, `addi $s24, $s0, 100`, `addi $s25, $s0, 100`, `addi $s26, $s0, 100`, `addi $s27, $s0, 100`, `addi $s28, $s0, 100`, `addi $s29, $s0, 100`, `addi $s30, $s0, 100`, `addi $s31, $s0, 100`, `addi $s32, $s0, 100`, `addi $s33, $s0, 100`, `addi $s34, $s0, 100`, `addi $s35, $s0, 100`, `addi $s36, $s0, 100`, `addi $s37, $s0, 100`, `addi $s38, $s0, 100`, `addi $s39, $s0, 100`, `addi $s40, $s0, 100`, `addi $s41, $s0, 100`, `addi $s42, $s0, 100`, `addi $s43, $s0, 100`, `addi $s44, $s0, 100`, `addi $s45, $s0, 100`, `addi $s46, $s0, 100`, `addi $s47, $s0, 100`, `addi $s48, $s0, 100`, `addi $s49, $s0, 100`, `addi $s50, $s0, 100`, `addi $s51, $s0, 100`, `addi $s52, $s0, 100`, `addi $s53, $s0, 100`, `addi $s54, $s0, 100`, `addi $s55, $s0, 100`, `addi $s56, $s0, 100`, `addi $s57, $s0, 100`, `addi $s58, $s0, 100`, `addi $s59, $s0, 100`, `addi $s60, $s0, 100`, `addi $s61, $s0, 100`, `addi $s62, $s0, 100`, `addi $s63, $s0, 100`, `addi $s64, $s0, 100`, `addi $s65, $s0, 100`, `addi $s66, $s0, 100`, `addi $s67, $s0, 100`, `addi $s68, $s0, 100`, `addi $s69, $s0, 100`, `addi $s70, $s0, 100`, `addi $s71, $s0, 100`, `addi $s72, $s0, 100`, `addi $s73, $s0, 100`, `addi $s74, $s0, 100`, `addi $s75, $s0, 100`, `addi $s76, $s0, 100`, `addi $s77, $s0, 100`, `addi $s78, $s0, 100`, `addi $s79, $s0, 100`, `addi $s80, $s0, 100`, `addi $s81, $s0, 100`, `addi $s82, $s0, 100`, `addi $s83, $s0, 100`, `addi $s84, $s0, 100`, `addi $s85, $s0, 100`, `addi $s86, $s0, 100`, `addi $s87, $s0, 100`, `addi $s88, $s0, 100`, `addi $s89, $s0, 100`, `addi $s90, $s0, 100`, `addi $s91, $s0, 100`, `addi $s92, $s0, 100`, `addi $s93, $s0, 100`, `addi $s94, $s0, 100`, `addi $s95, $s0, 100`, `addi $s96, $s0, 100`, `addi $s97, $s0, 100`, `addi $s98, $s0, 100`, `addi $s99, $s0, 100`, `addi $s100, $s0, 100`. On the right, there is a register window showing the values of registers \$0 through \$31. The values are mostly 0, with some non-zero values in the \$10 through \$31 range.

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

Challenge Problem:

Line: 3 Go!

Show/Hide Demos

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

Addition Doubler

Stav

Looper

Stack Test

Hello World

Code Gen Save String

Interactive

Binary2 Decimal

Decimal2 Binary

Debug

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

Step Run Enable auto switching

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

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

WeMIPS

Show/Hide Demos User Guide | Unit Tests | Docs

Addition Doubler Star Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

```
1 # Store 'hello world!' at the top of the stack
2 ADDI $a0, $zero, 32 # $0
3 SD $a0, 0($0)
4 ADDI $d0, $zero, 191 # e
5 SD $d0, 4($0)
6 SD $a0, 8($0)
7 ADDI $d0, $zero, 199 # l
8 SD $d0, 12($0)
9 ADDI $d0, $zero, 111 # o
10 SD $d0, 16($0)
11 ADDI $d0, $zero, 32 # (space)
12 SD $d0, 20($0)
13 ADDI $d0, $zero, 113 # w
14 SD $d0, 24($0)
15 ADDI $d0, $zero, 115 # a
16 SD $d0, 28($0)
17 ADDI $d0, $zero, 114 # r
18 SD $d0, 32($0)
19 ADDI $d0, $zero, 108 # l
20 SD $d0, 36($0)
21 ADDI $d0, $zero, 108 # l
22 SD $d0, 40($0)
23 ADDI $d0, $zero, 33 # d
24 SD $d0, 44($0)
25 ADDI $d0, $zero, 33 # l
26 SD $d0, 48($0)
27 ADDI $d0, $zero, 0 # (null)
28 SD $d0, 52($0)
29 #
30 ADDI $v0, $zero, 6 # 4 in for print string
31 ADDI $a0, $0, 0 # print to the log
32 syscall
```

Step	Run	Enable auto switching			
S	T	A	V	Stack	Log
a0				10	
a1				9	
a2				9	
a3				22	
a4				96	
a5				976	
a6				927	
a7				418	

(Demo with WeMIPS)

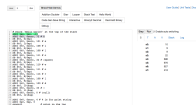
Today's Topics



- Design Patterns: Searching
- Python Recap
- Machine Language
- **Machine Language: Jumps & Loops**
- Binary & Hex Arithmetic
- 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.
- Can indicate locations by writing **labels** at the beginning of a line.



```
00100000: 0f 7f 00 00 mov     edi, 0
00100004: 31 d2          xor     edx, edx
00100006: 50          push  eax
00100007: 58          pop   eax
00100008: 8b 00          mov     eax, [eax]
0010000a: 8b 00          mov     eax, [eax]
0010000c: 8b 00          mov     eax, [eax]
0010000e: 8b 00          mov     eax, [eax]
00100010: 8b 00          mov     eax, [eax]
00100012: 8b 00          mov     eax, [eax]
00100014: 8b 00          mov     eax, [eax]
00100016: 8b 00          mov     eax, [eax]
00100018: 8b 00          mov     eax, [eax]
0010001a: 8b 00          mov     eax, [eax]
0010001c: 8b 00          mov     eax, [eax]
0010001e: 8b 00          mov     eax, [eax]
00100020: 8b 00          mov     eax, [eax]
00100022: 8b 00          mov     eax, [eax]
00100024: 8b 00          mov     eax, [eax]
00100026: 8b 00          mov     eax, [eax]
00100028: 8b 00          mov     eax, [eax]
0010002a: 8b 00          mov     eax, [eax]
0010002c: 8b 00          mov     eax, [eax]
0010002e: 8b 00          mov     eax, [eax]
00100030: 8b 00          mov     eax, [eax]
00100032: 8b 00          mov     eax, [eax]
00100034: 8b 00          mov     eax, [eax]
00100036: 8b 00          mov     eax, [eax]
00100038: 8b 00          mov     eax, [eax]
0010003a: 8b 00          mov     eax, [eax]
0010003c: 8b 00          mov     eax, [eax]
0010003e: 8b 00          mov     eax, [eax]
00100040: 8b 00          mov     eax, [eax]
00100042: 8b 00          mov     eax, [eax]
00100044: 8b 00          mov     eax, [eax]
00100046: 8b 00          mov     eax, [eax]
00100048: 8b 00          mov     eax, [eax]
0010004a: 8b 00          mov     eax, [eax]
0010004c: 8b 00          mov     eax, [eax]
0010004e: 8b 00          mov     eax, [eax]
00100050: 8b 00          mov     eax, [eax]
00100052: 8b 00          mov     eax, [eax]
00100054: 8b 00          mov     eax, [eax]
00100056: 8b 00          mov     eax, [eax]
00100058: 8b 00          mov     eax, [eax]
0010005a: 8b 00          mov     eax, [eax]
0010005c: 8b 00          mov     eax, [eax]
0010005e: 8b 00          mov     eax, [eax]
00100060: 8b 00          mov     eax, [eax]
00100062: 8b 00          mov     eax, [eax]
00100064: 8b 00          mov     eax, [eax]
00100066: 8b 00          mov     eax, [eax]
00100068: 8b 00          mov     eax, [eax]
0010006a: 8b 00          mov     eax, [eax]
0010006c: 8b 00          mov     eax, [eax]
0010006e: 8b 00          mov     eax, [eax]
00100070: 8b 00          mov     eax, [eax]
00100072: 8b 00          mov     eax, [eax]
00100074: 8b 00          mov     eax, [eax]
00100076: 8b 00          mov     eax, [eax]
00100078: 8b 00          mov     eax, [eax]
0010007a: 8b 00          mov     eax, [eax]
0010007c: 8b 00          mov     eax, [eax]
0010007e: 8b 00          mov     eax, [eax]
00100080: 8b 00          mov     eax, [eax]
00100082: 8b 00          mov     eax, [eax]
00100084: 8b 00          mov     eax, [eax]
00100086: 8b 00          mov     eax, [eax]
00100088: 8b 00          mov     eax, [eax]
0010008a: 8b 00          mov     eax, [eax]
0010008c: 8b 00          mov     eax, [eax]
0010008e: 8b 00          mov     eax, [eax]
00100090: 8b 00          mov     eax, [eax]
00100092: 8b 00          mov     eax, [eax]
00100094: 8b 00          mov     eax, [eax]
00100096: 8b 00          mov     eax, [eax]
00100098: 8b 00          mov     eax, [eax]
0010009a: 8b 00          mov     eax, [eax]
0010009c: 8b 00          mov     eax, [eax]
0010009e: 8b 00          mov     eax, [eax]
001000a0: 8b 00          mov     eax, [eax]
001000a2: 8b 00          mov     eax, [eax]
001000a4: 8b 00          mov     eax, [eax]
001000a6: 8b 00          mov     eax, [eax]
001000a8: 8b 00          mov     eax, [eax]
001000aa: 8b 00          mov     eax, [eax]
001000ac: 8b 00          mov     eax, [eax]
001000ae: 8b 00          mov     eax, [eax]
001000b0: 8b 00          mov     eax, [eax]
001000b2: 8b 00          mov     eax, [eax]
001000b4: 8b 00          mov     eax, [eax]
001000b6: 8b 00          mov     eax, [eax]
001000b8: 8b 00          mov     eax, [eax]
001000ba: 8b 00          mov     eax, [eax]
001000bc: 8b 00          mov     eax, [eax]
001000be: 8b 00          mov     eax, [eax]
001000c0: 8b 00          mov     eax, [eax]
001000c2: 8b 00          mov     eax, [eax]
001000c4: 8b 00          mov     eax, [eax]
001000c6: 8b 00          mov     eax, [eax]
001000c8: 8b 00          mov     eax, [eax]
001000ca: 8b 00          mov     eax, [eax]
001000cc: 8b 00          mov     eax, [eax]
001000ce: 8b 00          mov     eax, [eax]
001000d0: 8b 00          mov     eax, [eax]
001000d2: 8b 00          mov     eax, [eax]
001000d4: 8b 00          mov     eax, [eax]
001000d6: 8b 00          mov     eax, [eax]
001000d8: 8b 00          mov     eax, [eax]
001000da: 8b 00          mov     eax, [eax]
001000dc: 8b 00          mov     eax, [eax]
001000de: 8b 00          mov     eax, [eax]
001000e0: 8b 00          mov     eax, [eax]
001000e2: 8b 00          mov     eax, [eax]
001000e4: 8b 00          mov     eax, [eax]
001000e6: 8b 00          mov     eax, [eax]
001000e8: 8b 00          mov     eax, [eax]
001000ea: 8b 00          mov     eax, [eax]
001000ec: 8b 00          mov     eax, [eax]
001000ee: 8b 00          mov     eax, [eax]
001000f0: 8b 00          mov     eax, [eax]
001000f2: 8b 00          mov     eax, [eax]
001000f4: 8b 00          mov     eax, [eax]
001000f6: 8b 00          mov     eax, [eax]
001000f8: 8b 00          mov     eax, [eax]
001000fa: 8b 00          mov     eax, [eax]
001000fc: 8b 00          mov     eax, [eax]
001000fe: 8b 00          mov     eax, [eax]
```


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:



The screenshot shows a debugger window with two panes. The left pane displays assembly code with several lines of instructions, including labels like `loop_start` and `loop_end`. The right pane shows a disassembled instruction, likely a jump instruction, with its corresponding machine code and hex values.

Loops & Jumps in Machine Language

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



The screenshot shows a debugger window with two panes. The left pane displays assembly code with labels like `start` and `done`. The right pane shows a disassembled instruction, likely a jump instruction, with its address and parameters.

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.



```

# Example 1.1: A simple loop in assembly language
# This program calculates the sum of integers from 1 to 10.
# It uses a loop to add each integer to a running total.

# Register $s0 holds the current integer to be added.
# Register $s1 holds the running total.

# Initialize $s0 to 1 and $s1 to 0.
li $s0, 1
li $s1, 0

# Loop: Add $s0 to $s1, then increment $s0.
loop:
add $s1, $s1, $s0
addi $s0, $s0, 1

# Jump back to the start of the loop.
j loop

# End of program.

```

Loops & Jumps in Machine Language

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



```

# Example 1.1: A simple loop in assembly
# This program prints the number 1 through 10.

# Set up registers
li $t0, 1          # $t0 = 1
li $t1, 10         # $t1 = 10

loop:              # Label: loop
    printd $t0     # Print $t0
    addi $t0, $t0, 1  # $t0 = $t0 + 1
    bne $t0, $t1, done  # If $t0 != $t1, jump to done
    j loop         # Jump back to loop

done:              # Label: done
    # End of program

```

Jump Demo

Line 3 | dis

ShowHide Demos

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

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

Debug

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

User Guide | Unit Tests | Docs

Step | Run | Enable auto switching

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

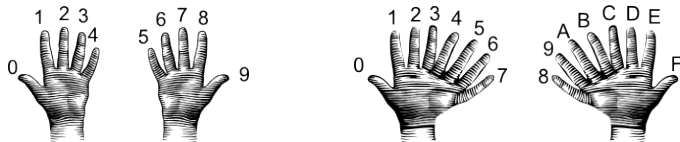
(Demo with WeMIPS)

Today's Topics



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

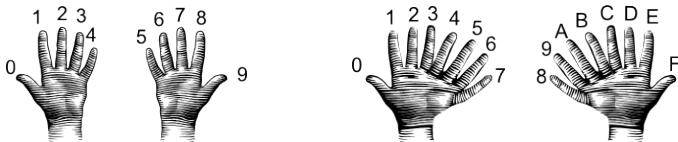
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

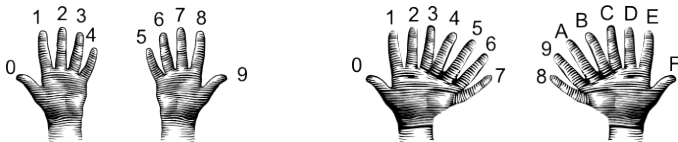
Hexadecimal to Decimal: Converting Between Bases



(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.

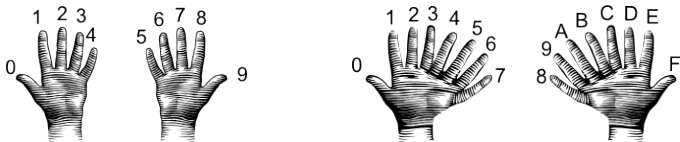
Hexadecimal to Decimal: Converting Between Bases



(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?

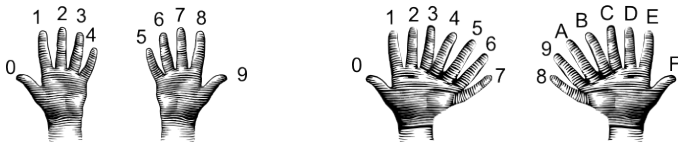
Hexadecimal to Decimal: Converting Between Bases



(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

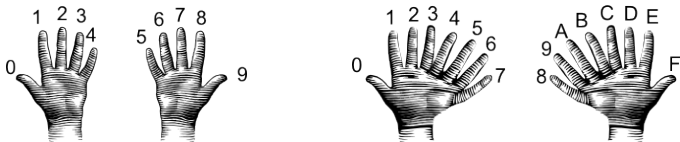


(from i-programmer.info)

- 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. 2×16 is 32.

Hexadecimal to Decimal: Converting Between Bases

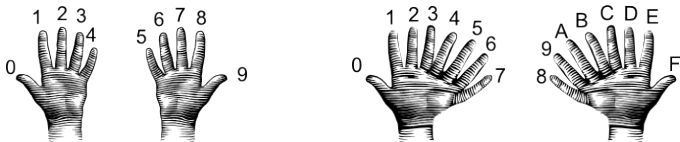


(from i-programmer.info)

- 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. 2×16 is 32.
A in decimal digits is 10.

Hexadecimal to Decimal: Converting Between Bases

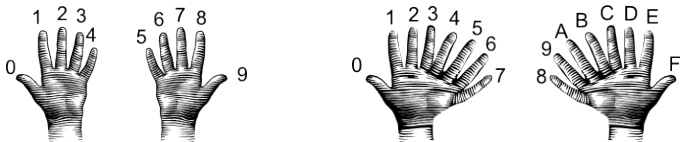


(from i-programmer.info)

- 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. 2×16 is 32.
A in decimal digits is 10.
 $32 + 10$ is 42.

Hexadecimal to Decimal: Converting Between Bases

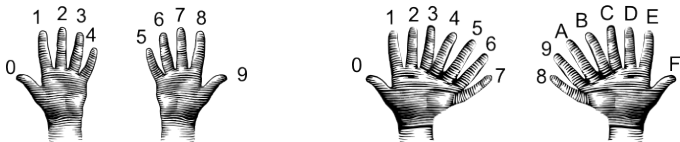


(from i-programmer.info)

- 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. 2×16 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

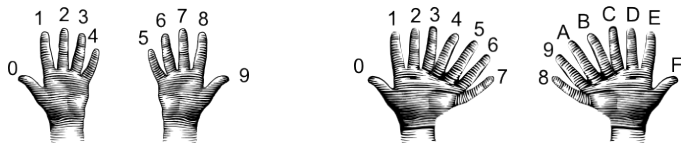


(from i-programmer.info)

- 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. 2×16 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

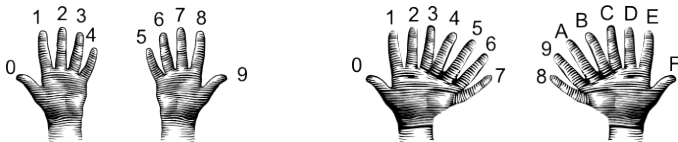


(from i-programmer.info)

- 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. 2×16 is 32.
A in decimal digits is 10.
 $32 + 10$ is 42.
Answer is 42.
- ▶ Example: what is 99 as a decimal number?
9 in decimal is 9. 9×16 is 144.

Hexadecimal to Decimal: Converting Between Bases

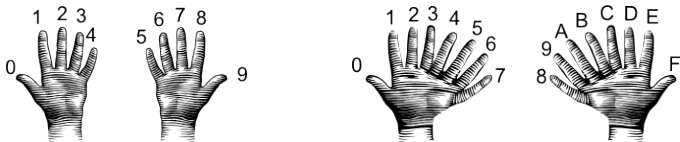


(from i-programmer.info)

- From hexadecimal to decimal:

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

Hexadecimal to Decimal: Converting Between Bases

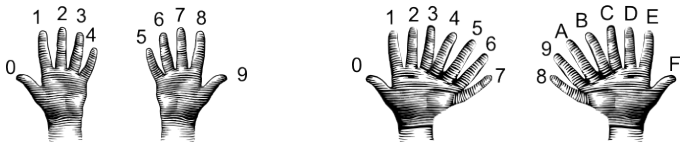


(from i-programmer.info)

- From hexadecimal to decimal:

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?
2 in decimal is 2. 2×16 is 32.
A in decimal digits is 10.
 $32 + 10$ is 42.
Answer is 42.
- ▶ Example: what is 99 as a decimal number?
9 in decimal is 9. 9×16 is 144.
9 in decimal digits is 9
 $144 + 9$ is 153.

Hexadecimal to Decimal: Converting Between Bases

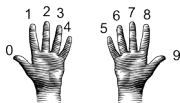


(from i-programmer.info)

- From hexadecimal to decimal:

- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?
2 in decimal is 2. 2×16 is 32.
A in decimal digits is 10.
 $32 + 10$ is 42.
Answer is 42.
- ▶ Example: what is 99 as a decimal number?
9 in decimal is 9. 9×16 is 144.
9 in decimal digits is 9
 $144 + 9$ is 153.
Answer is 153.

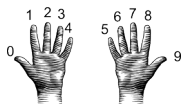
Decimal to Binary: Converting Between Bases



- From decimal to binary:

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

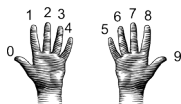
Decimal to Binary: Converting Between Bases



- 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.

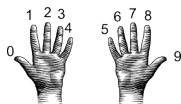
Decimal to Binary: Converting Between Bases



- 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.

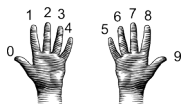
Decimal to Binary: Converting Between Bases



- 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.

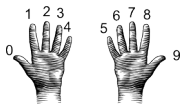
Decimal to Binary: Converting Between Bases



- 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.

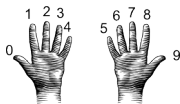
Decimal to Binary: Converting Between Bases



- 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.

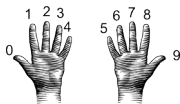
Decimal to Binary: Converting Between Bases



- 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.

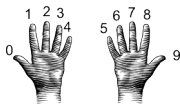
Decimal to Binary: Converting Between Bases



- 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.

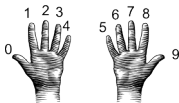
Decimal to Binary: Converting Between Bases



- 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

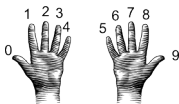


- From decimal to binary:

- ▶ 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

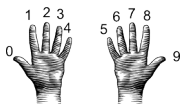


- From decimal to binary:

- ▶ 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



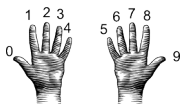
- From decimal to binary:

- ▶ 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



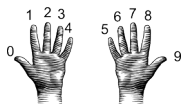
- From decimal to binary:

- ▶ 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



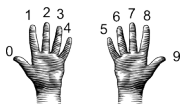
- From decimal to binary:

- ▶ 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:

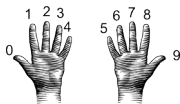
- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient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

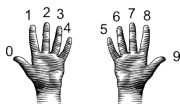
- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient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

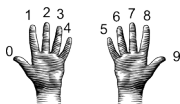
- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient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

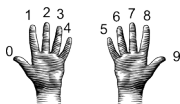


- From decimal to binary:

- ▶ 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?

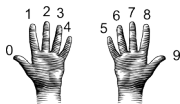
130/128 is 1 rem 2. First digit is 1: 1...

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

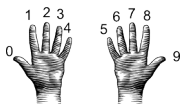


● From decimal to binary:

- ▶ 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

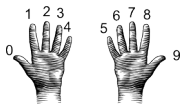


- From decimal to binary:

- ▶ 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

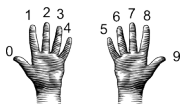


- From decimal to binary:

- ▶ 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

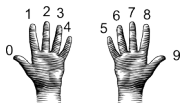


- From decimal to binary:

- ▶ 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

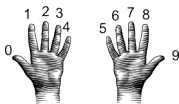


- From decimal to binary:

- ▶ 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

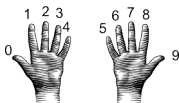


- From decimal to binary:

- ▶ 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

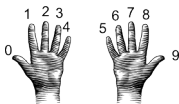


- From decimal to binary:

- ▶ 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

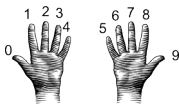


- From decimal to binary:

- ▶ 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

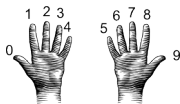


- From decimal to binary:

- ▶ 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

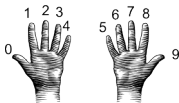


- From decimal to binary:

- ▶ 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

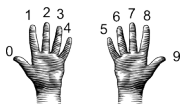


- From decimal to binary:

- ▶ 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

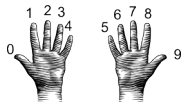


- From decimal to binary:

- ▶ 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?

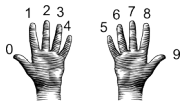
```
130/128 is 1 rem 2. First digit is 1: 1...
2/64 is 0 rem 2. Next digit is 0: 10...
2/32 is 0 rem 2. Next digit is 0: 100...
2/16 is 0 rem 2. Next digit is 0: 1000...
2/8 is 0 rem 2. Next digit is 0: 10000...
2/4 is 0 remainder 2. Next digit is 0: 100000...
2/2 is 1 rem 0. Next digit is 1: 1000001...
Adding the last remainder: 10000010
```

Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

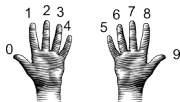
Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

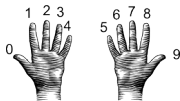
99/128 is 0 rem 99.

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

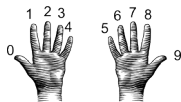


- Example: what is 99 in binary notation?

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

99/64 is 1 rem 35.

Decimal to Binary: Converting Between Bases

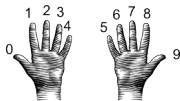


- Example: what is 99 in binary notation?

$99/128$ 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

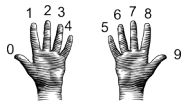


- Example: what is 99 in binary notation?

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



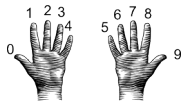
- Example: what is 99 in binary notation?

$99/128$ 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



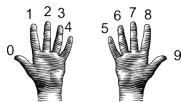
- Example: what is 99 in binary notation?

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



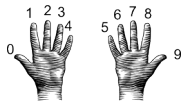
- Example: what is 99 in binary notation?

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?

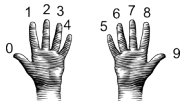
$99/128$ is 0 rem 99. First digit is 0: 0...

$99/64$ 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?

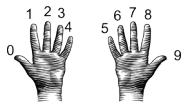
99/128 is 0 rem 99. First digit is 0: 0...

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:

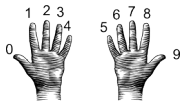
Decimal to Binary: Converting Between Bases



- 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...

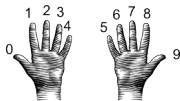
Decimal to Binary: Converting Between Bases



- 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.

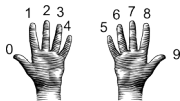
Decimal to Binary: Converting Between Bases



- 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:

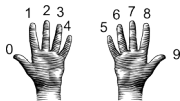
Decimal to Binary: Converting Between Bases



- 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...

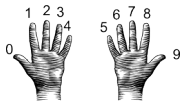
35/32 is 1 rem 3. Next digit is 1: 011...

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...

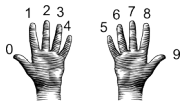
35/32 is 1 rem 3. Next digit is 1: 011...

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...

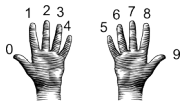
35/32 is 1 rem 3. Next digit is 1: 011...

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...

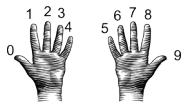
Decimal to Binary: Converting Between Bases



- 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...

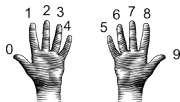
3/16 is 0 rem 3. Next digit is 0: 0110...

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:

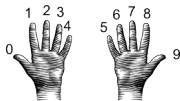
Decimal to Binary: Converting Between Bases



- 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 rem 3.  Next digit is 0:     011000...
3/2    is 1 rem 1.  Next digit is 1:     0110001...
```

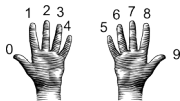
Decimal to Binary: Converting Between Bases



- 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

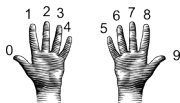


- Example: what is 99 in binary notation?

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

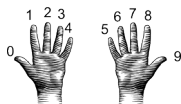
Answer is 1100011.

Binary to Decimal: Converting Between Bases



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

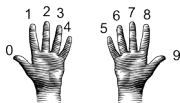
Binary to Decimal: Converting Between Bases



- From binary to decimal:

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

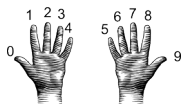
Binary to Decimal: Converting Between Bases



- 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.

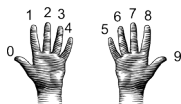
Binary to Decimal: Converting Between Bases



- From binary to decimal:

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

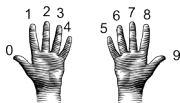
Binary to Decimal: Converting Between Bases



- From binary to decimal:

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

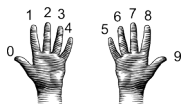
Binary to Decimal: Converting Between Bases



- From binary to decimal:

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

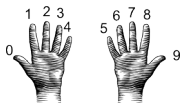
Binary to Decimal: Converting Between Bases



- From binary to decimal:

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

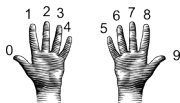
Binary to Decimal: Converting Between Bases



- From binary to decimal:

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

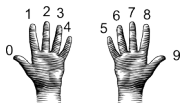
Binary to Decimal: Converting Between Bases



- From binary to decimal:

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

Binary to Decimal: Converting Between Bases

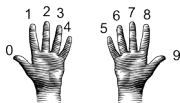


● From binary to decimal:

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

Sum starts with:

Binary to Decimal: Converting Between Bases



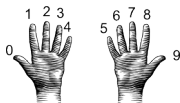
● From binary to decimal:

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

Sum starts with: 1

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

Binary to Decimal: Converting Between Bases

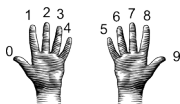


- From binary to decimal:

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

Binary to Decimal: Converting Between Bases

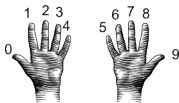


- From binary to decimal:

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

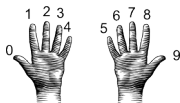


● From binary to decimal:

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

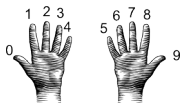


● From binary to decimal:

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

Binary to Decimal: Converting Between Bases

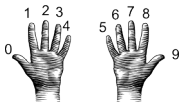


● From binary to decimal:

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

Binary to Decimal: Converting Between Bases

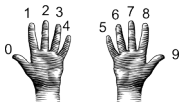


● From binary to decimal:

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

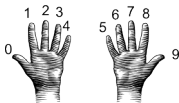


● From binary to decimal:

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

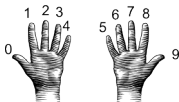


● From binary to decimal:

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

Binary to Decimal: Converting Between Bases

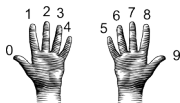


- From binary to decimal:

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

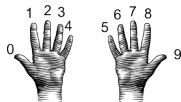


- From binary to decimal:

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

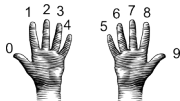
Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with:

Binary to Decimal: Converting Between Bases

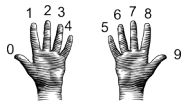


- Example: What is 10100100 in decimal?

Sum starts with: 0

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

Binary to Decimal: Converting Between Bases

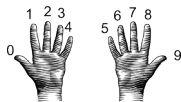


- Example: What is 10100100 in decimal?

Sum starts with: 0

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

Binary to Decimal: Converting Between Bases



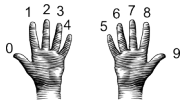
- Example: What is 10100100 in decimal?

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



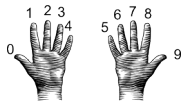
- Example: What is 10100100 in decimal?

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?

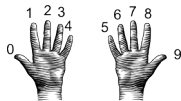
Sum starts with: 0

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

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

$0 \cdot 8 = 0$. Add 0 to sum:

Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

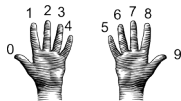
Sum starts with: 0

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

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

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

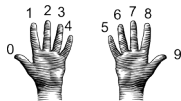
Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

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

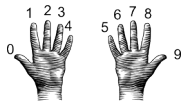
Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

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

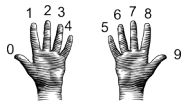
Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

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

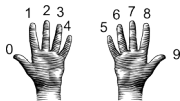
Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

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

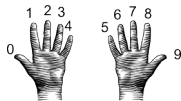
Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

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

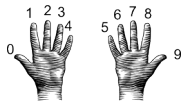
Binary to Decimal: Converting Between Bases



- 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

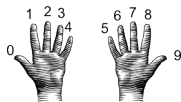
Binary to Decimal: Converting Between Bases



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

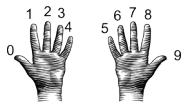
Binary to Decimal: Converting Between Bases



- 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

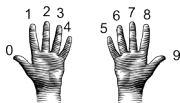


- Example: What is 10100100 in decimal?

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

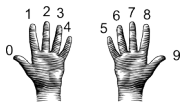
The answer is 164.

Design Challenge: Incrementers



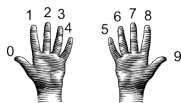
- Simplest arithmetic: add one (“increment”) a variable.

Design Challenge: Incrementers



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

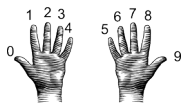
Design Challenge: Incrementers



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

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

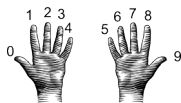

Design Challenge: Incrementers



- 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.

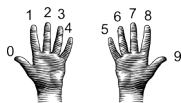
Design Challenge: Incrementers



- 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"

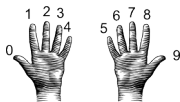
Design Challenge: Incrementers



- 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.

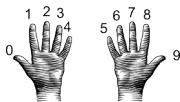
Design Challenge: Incrementers



- 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



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

Recap



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

Recap



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

Today's Topics



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

Final Overview: Administration

- The exam will be administered through Gradescope.

Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on **the final day of class 9:50a - 12p**

Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on **the final day of class 9:50a - 12p**
- The morning of the exam: log into Gradescope, find the **CSci 127 Final Exam** and open the assignment.
- Final Exam will resember quizzes administered through out the semester

Final Overview: Format

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

Final Overview: Format

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

Final Overview: Format

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

Final Overview: Format

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

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
 - ▶ 10 questions, each worth 10 points.

Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
 - ▶ With notes, examples, programs: what will help you on the exam.
 - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
 - ▶ 10 questions, each worth 10 points.
 - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.

Final Overview: Format

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

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