

# CSci 127: Introduction to Computer Science



[hunter.cuny.edu/csci](http://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: 9 Classes in 10 Minutes



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

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

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

- Introduced comments & print():

```
#Name: Thomas Hunter
```

← These lines are comments

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

← (for us, not computer to read)

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

← (this one also)

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

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

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

- Introduced comments & print():

```
#Name: Thomas Hunter
```

← These lines are comments

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

← (for us, not computer to read)

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

← (this one also)

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

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

- As well as definite loops & the turtle package:

The screenshot shows a Python code editor interface. On the left, the code file 'main.py' is open, containing 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)
```

On the right, the 'Result' tab displays the output of the program: a purple hexagon drawn on the screen with six star-shaped markers at each vertex, representing turtle stamps.

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

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

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

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

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers

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

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers

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

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers
  - ▶ **string**: sequence of characters

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

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers
  - ▶ **string**: sequence of characters
  - ▶ **list**: a sequence of items

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

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

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

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

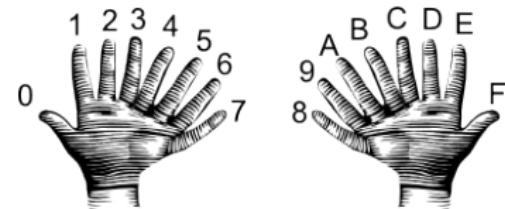
# Class 1: variables, data types, more on loops & range()

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

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

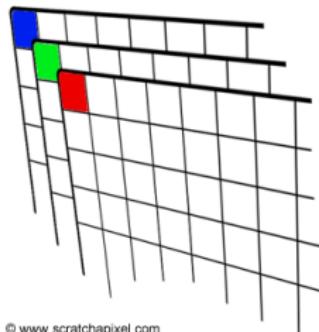
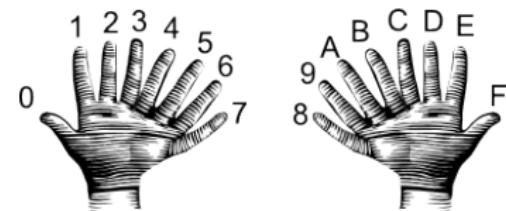
# Class 2: colors, hex, slices, numpy & images

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



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

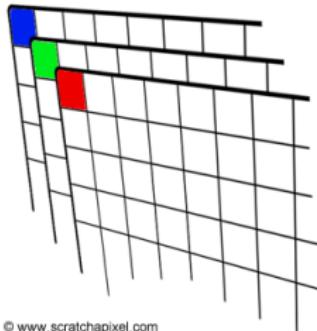
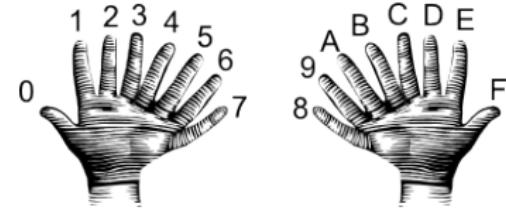
Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



© www.scratchapixel.com

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

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



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

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

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

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

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

# Class 3: design problem (cropping images) & decisions



# Class 3: design problem (cropping images) & decisions



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

# Class 3: design problem (cropping images) & decisions



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

# Class 3: design problem (cropping images) & decisions



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

## Class 3: design problem (cropping images) & decisions

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

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

# Class 4: logical operators, truth tables & logical circuits

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

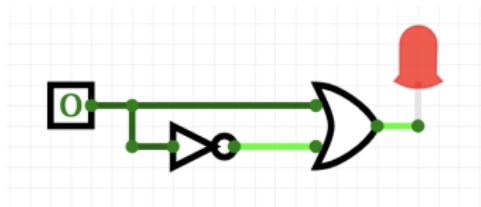
visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
    (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

# Class 4: logical operators, truth tables & logical circuits

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

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

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



# Class 5: structured data, pandas, & more design

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

.....  
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total  
1690,4937,2017,...,727,7881  
1771,21843,36232,...,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,67541,6240,6842,1755,4543,79334  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,16013,14034,5346,10965,391114  
1850,35344,128912,189511,148915,5346,10965,391115  
1860,613469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33029,1911801  
1890,1364473,65943,5653,51980,33029,1911801  
1900,185093,116582,152999,200567,67621,2437202  
1910,2233142,1634351,2841,430980,8569,4766803  
1920,22331103,2018354,44601,430980,73201,11651,59148  
1930,1667111,1297128,1297128,1297128,1297128,4930446  
1940,1889924,2498285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7891957  
1960,1696101,2738175,1550949,1451277,191555,7891957  
1970,1696101,2738175,1550949,1451277,191555,7891984  
1970,1696101,2738175,1550949,1451277,191555,7891984  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1203789,378977,7322564  
2000,1537195,2485326,2229379,1332450,419782,8080879  
2010,1583873,2504705,2277722,1385108,447558,8155133  
2015,1444018,2646733,2339150,1454446,474558,8059405

nycHistPop.csv

In Lab 6

# Class 5: structured data, pandas, & more design

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

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

.....  
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total  
1890,4937,2037,,727,7881,28423  
1771,21843,36231,,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75935  
1810,67541,6240,6842,1755,4543,75934  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,311510,19013,14045,5346,10965,391114  
1850,355441,218913,18895,5346,10965,415115  
1860,813449,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33029,1911801  
1890,1367111,70000,6540,51980,33029,2148134  
1900,185093,116582,152999,200567,67621,2437202  
1910,223342,1634351,284641,430980,8569,476683  
1920,2211103,2018354,446071,446071,72021,116515,50048  
1930,1867132,1634351,284641,430980,8569,476683  
1940,1889924,2498285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690101,2738175,1550949,1451277,191555,7981984  
1970,1539231,2460712,1472121,1472121,135443,7981984  
1980,1426285,2230936,1891325,1168972,252121,7071639  
1990,1487536,2300664,1951598,1203789,737977,7232564  
2000,1537195,2485326,2223379,1332450,419728,8080879  
2010,1583873,2504705,2272722,1385108,8175133  
2015,1444518,2436733,2339150,1459444,474558,8059405

nycHistPop.csv

In Lab 6

# Class 5: structured data, pandas, & more design

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

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

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City.....  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....  
.....  
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total  
1690,4937,2037,,727,7181,12000  
1771,21843,36231,,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75935  
1810,68000,62000,68000,18000,45000,103034  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,30323,7082,242278  
1840,312110,18013,14000,5346,10965,391114  
1850,355400,218000,185000,58500,115000,500000  
1860,813469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,599403,56537,51980,33091,1911803  
1890,1380000,720000,680000,550000,318000,3800000  
1900,1850993,116582,152999,200567,67621,2437202  
1910,2233142,1634351,28441,430980,8569,476683  
1920,22101103,2018354,44600,720201,116700,5000000  
1930,18671100,1790000,1790000,1790000,1790000,4930446  
1940,1889924,2498285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690000,2000000,1800000,1800000,1800000,6781984  
1970,1539231,2460701,2147201,2135443,7091846  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1302789,737977,7322564  
2000,1537195,2485326,2229379,1332650,419728,8080879  
2010,1583873,2504705,2216722,1385108,8175133,8175133  
2015,1444018,2642733,2339150,1459446,474558,8059405
```

nycHistPop.csv

In Lab 6

# Class 5: structured data, pandas, & more design

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

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

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City.....  
All population figures are consistent with present-day boundaries.....  
Five census after the consolidation of the five boroughs.....  
.....  
Year,Population  
1690,4937,2017,...,727,718120  
1771,21843,36231,...,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,70000,62000,68000,18000,49734  
1820,123704,11487,8246,2792,6135,152056  
1830,20589,20535,9049,30323,7082,242278  
1840,31510,19113,14000,5348,10965,391114  
1850,35540,21800,18500,5800,12000,45115  
1860,813469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59940,5653,51980,33091,1911801  
1890,1360000,720000,68000,51800,31800,1911814  
1900,1850993,1165852,152999,200567,67921,2437202  
1910,2233142,1634351,2841,430980,8569,4766883  
1920,22161103,2018354,44600,72021,11651,50048  
1930,18671128,1796128,1796128,135254,15831,4930446  
1940,1889924,2498285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690000,2319319,1809000,1400000,1809000,781984  
1970,1539231,2465070,1874731,1472701,135443,768460  
1980,1428285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1302789,737977,7322564  
2000,1537195,2485326,2229379,1332650,419782,8080879  
2010,1583873,2504705,2277722,1385108,8175133  
2015,1444518,2436733,2339150,1459446,474558,8056405
```

nycHistPop.csv

In Lab 6

# Class 5: structured data, pandas, & more design

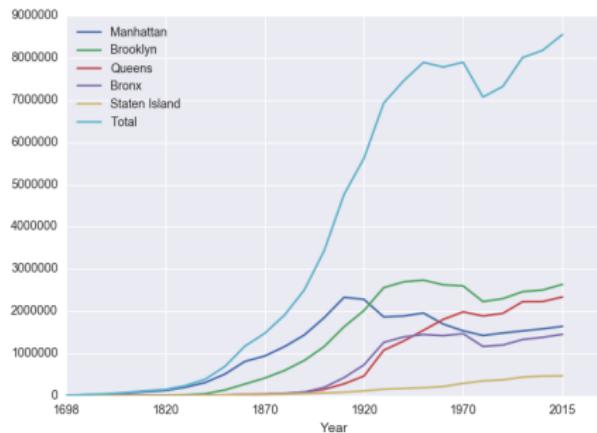
```
import matplotlib.pyplot as plt  
import pandas as pd
```

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

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City.....  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....  
.....  
Year,Borough,Population  
1699,Manhattan, Brooklyn, Queens, Bronx, Staten Island, Total  
1771,21843,36231,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75935  
1810,70531,6354,7041,1811,4973,93734  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,18013,14041,5348,10965,391114  
1850,455441,21800,18851,6851,11545,51154  
1860,813469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33051,1911801  
1890,1400000,720000,680000,518000,350000,200000  
1900,1850093,1165852,152999,200567,67921,2437202  
1910,2331542,1634351,2841,430980,8569,476683  
1920,2281103,2018354,44601,732013,11651,50048  
1930,2667123,2407128,35254,52545,15837,6930446  
1940,188924,2698285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550849,1451277,191555,7991957  
1960,1696000,2303219,1809000,1460000,1300000,781984  
1970,1539231,2465071,1472701,1235443,798462  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1320789,378977,7322564  
2000,1537195,2485326,2229379,1332650,413728,8080879  
2010,1583873,2504705,2216722,1385108,415712,8175133  
2015,1444518,2636733,2339150,1459446,474558,8059405
```

nycHistPop.csv

In Lab 6



# Class 6: functions

- Functions are a way to break code into pieces, that can be easily reused.

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

# Class 6: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.

# Class 6: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`

# Class 6: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:

# Class 6: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:  
Example: `print("Hello", "World")`

# Class 6: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:  
Example: `print("Hello", "World")`
- Can write, or **define** your own functions,

# Class 6: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:  
Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked or called.

# Class 7: function parameters, github

- Functions can have **input parameters**.

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

# Class 7: function parameters, github

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

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

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

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

# Class 7: function parameters, github

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

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

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

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

# Class 7: function parameters, github

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

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

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

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

# Class 7: function parameters, github

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

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

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

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

# Class 7: function parameters, github

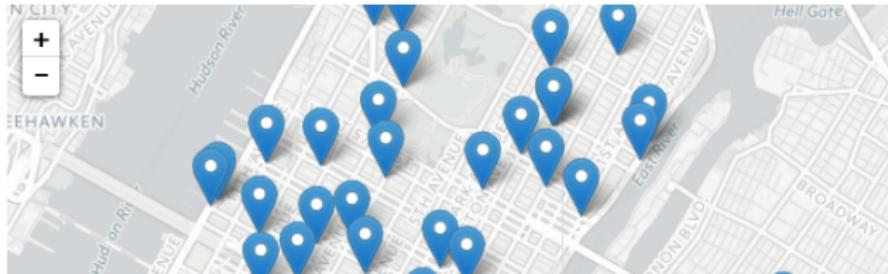
```
def totalWithTax(food, tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

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

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

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

## Class 8: top-down design, folium, loops, and random()



```
def main():
    dataF = getData()
    latColName, lonColName = getColumnNames()
    lat, lon = getLocale()
    cityMap = folium.Map(location = [lat,lon], tiles = 'cartodbpositron',zoom_start=11)
    dotAllPoints(cityMap,dataF,latColName,lonColName)
    markAndFindClosest(cityMap,dataF,latColName,lonColName,lat,lon)
    writeMap(cityMap)
```

## Class 9: more on loops, max design pattern, random()

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

print('The distance entered is', dist)
```

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

```
import turtle
import random

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

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

## Class 9: more on loops, max design pattern, random()

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

print('The distance entered is', dist)
```

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

```
import turtle
import random

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

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

## Class 9: more on loops, max design pattern, random()

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

print('The distance entered is', dist)
```

```
import turtle
import random

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

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

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

## Class 9: more on loops, max design pattern, random()

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

print('The distance entered is', dist)
```

```
import turtle
import random

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

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

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

## Class 9: more on loops, max design pattern, random()

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

print('The distance entered is', dist)
```

```
import turtle
import random

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

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

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

# Python & Circuits Review: 9 Classes in 10 Minutes



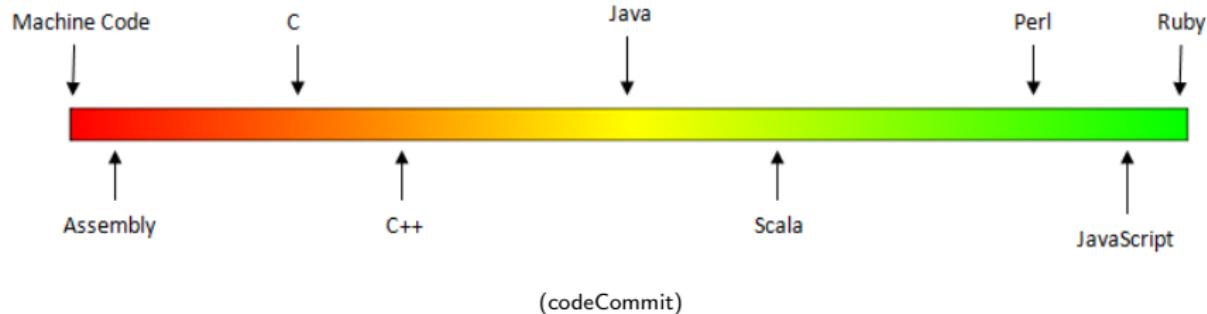
- Input/Output (I/O): `input()` and `print()`; pandas for CSV files
- Types:
  - ▶ Primitive: `int`, `float`, `bool`, `string`;
  - ▶ Container: lists (but not dictionaries/hashes or tuples)
- Objects: turtles (used but did not design our own)
- Loops: definite & indefinite
- Conditionals: `if-elif-else`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

# Today's Topics



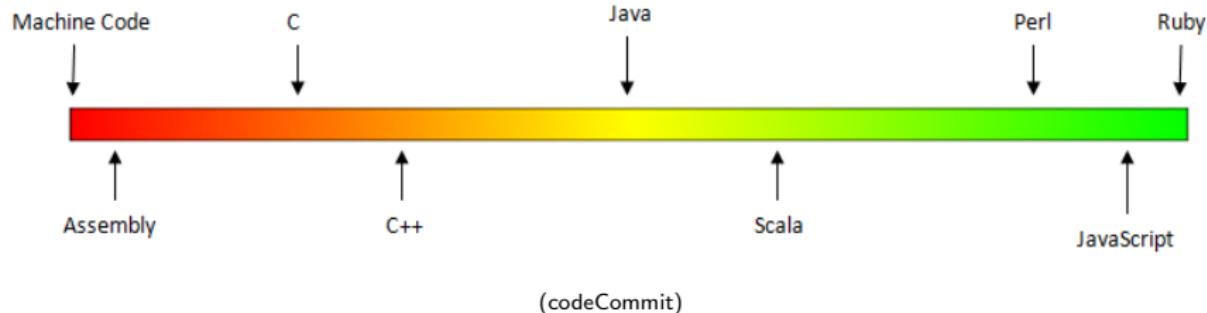
- Design Patterns: Searching
- Python Recap
- **Machine Language**
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

# Low-Level vs. High-Level Languages



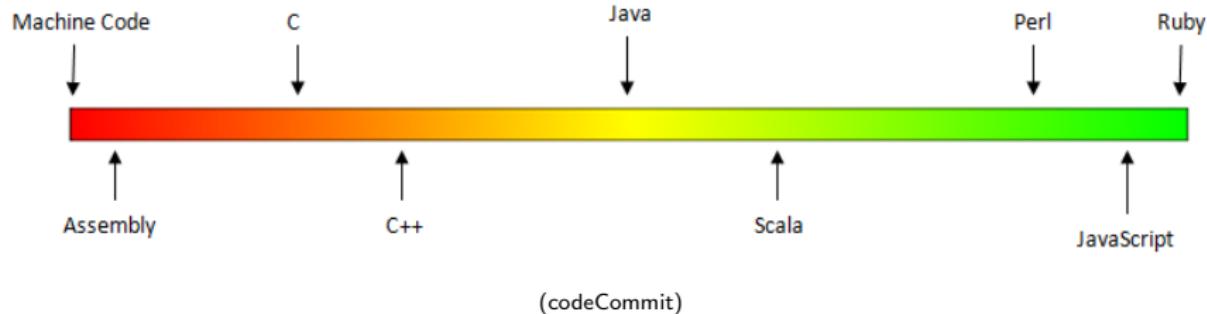
- Can view programming languages on a continuum.

# Low-Level vs. High-Level Languages



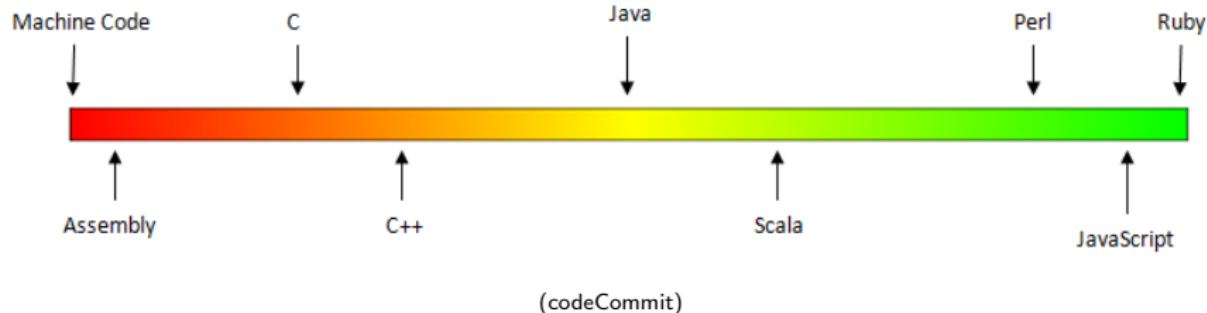
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

# Low-Level vs. High-Level Languages



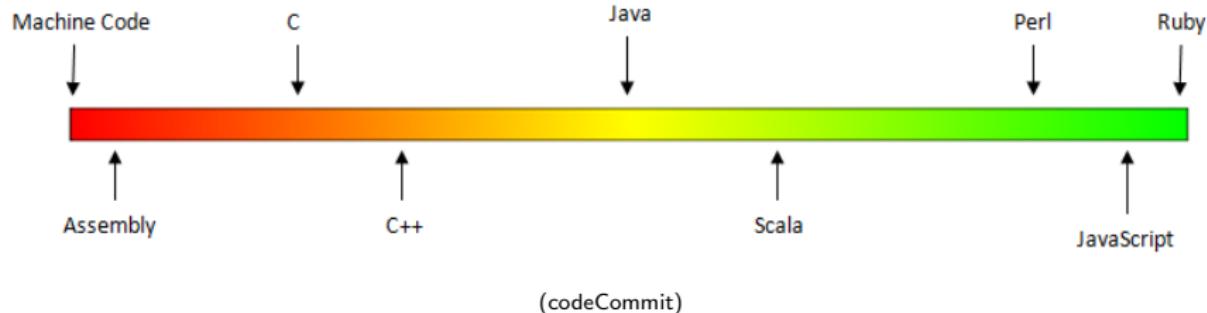
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

# 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.

# Processing

Blindtext: Ein Blindtext ist ein Text, der auf den ersten Blick nicht als solcher erkannt werden kann. Er besteht aus einer unleserlichen Abfolge von Zeichen, die durch spezielle Verarbeitung (z.B. mit einem Computerprogramm) entzifferbar sind.

Dies ist ein Blindtext. An ihm lässt sich vieles über die Schrift ableSEN. An der geSETZT ist. Auf den ersten BLICK wird der GRASWURZEL der Schriftfläche SICHTBAR. Dann kann man prüfen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt.  
Dies ist ein Blindtext. An ihm lässt sich



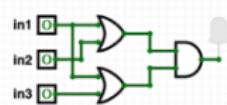
Data  
&  
Instructions

```
0110100011100100110000101  
1100100011011011011011011  
00111100001101000101011  
001011010000100000111111  
11010111101000111011011  
00100101010110011000000  
01001011010101001000000  
11100100000110100110101  
011000101101100011010101  
010000100000001100010000  
0110010101100110010101  
10010100000011011001101  
01101100100010001000000  
01100011011011001000000  
011001101101101101101100  
0110011000110000001101  
0110011000000010001101  
001000001110100011011100  
000101000110100011011100  
01101000110111011011000  
1101011011001101001001  
011100100101110010001101  
000000110110001101100011
```

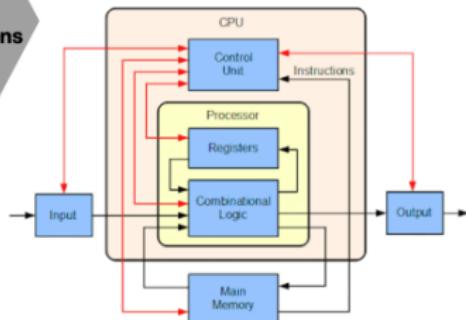


```
def totalWithTax(Food,tip):  
    total = 8  
    tax = 0.0875  
    total = Food + food * tax  
    total = total + tip  
    return(total)
```

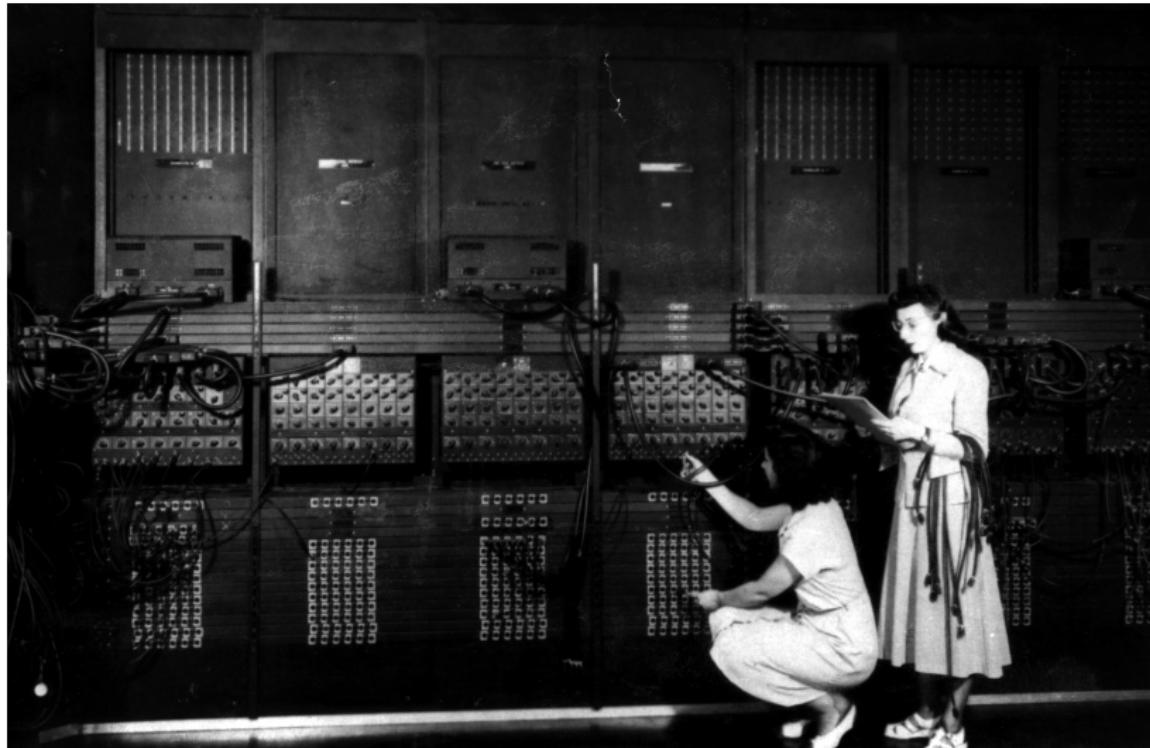
Data  
&  
Instructions



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



# Machine Language



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

# Machine Language

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

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

BREAK

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

(wiki)

# Machine Language

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



The screenshot shows a terminal window with assembly code and register values. The assembly code is:

```
R 00200000 C2 30 [4:1]a 21-  
R 00200002 1B C0  
R 00200003 FB SEI  
R 00200004 69 34 12 LDA #1234  
R 00200007 69 21 43 LDH #4321  
R 00200008 8F 03 7F 01 STA #17F03  
R 0020000E 8F E2 30 CLD  
R 0020000F E2 30 SEC  
R 00200011 80 SEK  
R 00200012  
P
```

Registers (PC, R0-R7, SP, DP, flags):

PC	MJWx012C	A	X	Y	SP	DP	RR
: 00	E012	00110000	0000	0000	0002	CFFF	0000 00
& 2000							

BREAK

Registers (PC, R0-R7, SP, DP, flags):

PC	MJWx012C	A	X	Y	SP	DP	RR
: 00	E012	00110000	5555	0000	0002	CFFF	0000 00
& 2000	7F03 7F03	007FFF	03 55	00 00	00 00	00 00	00 00 00 00 00 00 00 00

(wiki)

# Machine Language

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

A screenshot of a computer screen displaying a WeMIPS assembly editor. The top half shows assembly code with comments and labels. The bottom half shows the corresponding binary machine code and register values. The assembly code includes instructions like REP, LD, SD, ST, CLD, and SEV. The binary output shows fields for PC, MULc, A, X, Y, SP, DP, and various immediate values. A 'BREAK.' command is present.

```
    .C2 38        REP    $#38
    .0000021B    CLD
    .000003FB    SD
    .00000450    LDA    #1234
    .00000769 2143    LDCA #4324
    .000008BF 037F 01    STW    #017F03
    .000009E8    CLD
    .00000AE2 30    SEV    #38
    .00001180    BRK
    .000012
; PB PC    MULc 312C  A   X   Y   SP   DP   R
; 00 E912 00100000 0000 0000 0002 CFFF 0000 00
& 2000
BREAK.
PB PC    MULc 312C  A   X   Y   SP   DP   R
A 00 E912 00100000 0000 0000 0002 CFFF 0000 00
& 2000
7703 2703 >007FF0 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

(wiki)

# Machine Language



The screenshot shows a terminal window with two tabs open. The left tab displays assembly language code:

```
R 002800 C2 3B REP #830
R 002802 1B CLD
R 002803 FB SEI
R 002804 69 34 12 LDA #1234
R 002807 69 21 43 LDC #4321
R 002808 8F 03 7F 01 STA #017F03
R 00280E E9 30 CLD
R 002811 80 SEI #38
R 002812 00 BRK
R 2012
```

The right tab shows a memory dump with columns for Address, PC, Value, and Registers (A, X, Y, SP, DP, IR, SPK). The dump includes the instruction at address R 002800 and the stack area starting at R 2000.

Address	PC	Value	A	X	Y	SP	DP	IR	SPK
R 002800	002800	00100000 0000 0000 0002 CFFF 0000 00							
R 002802	002802	00100000 5555 0000 0002 CFFF 0000 00							
R 002803	002803	00100000 5555 0000 0000 0000 0000 0000 00							
R 2000	2000	00100000 0000 0000 0000 0000 0000 0000 00							

(wiki)

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

# Machine Language



The screenshot shows a terminal window with assembly code and its corresponding binary output. The assembly code includes instructions like REP, LD, ST, CLD, and SEV. Below the assembly code, the binary representation is shown in columns for PC, M[PC], A, X, Y, SP, DP, and R. The bottom part of the window displays the word 'BREAK'.

```
R:00200000 C2 30 [1:1]a 27-  
R:0020002 1B CD  
R:0020003 FB CD  
R:0020004 69 34 12 LDA #1234  
R:0020007 69 21 43 LDC #4321  
R:0020008 8F 03 7F 01 STA #17F03  
R:002000E E9 30 CLD  
R:0020010 69 38 SEV #38  
R:0020011 80 BRK  
R:0020012  
  
PB PC M[PC] A X Y SP DP R  
: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
& 20000  
  
BREAK
```

(wiki)

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

# "Hello World!" in Simplified Machine Language

Line: 3 Go!

Show/Hide Demos

User Guide | Unit Tests | Docs

Addition Doubler Stav Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

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

Step

Run

Enable auto switching

S

T

A

V

Stack

Log

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

(WeMIPS)



# WeMIPS

The screenshot shows the WeMIPS IDE interface. At the top, there are tabs for 'Run' (selected), 'Data', and 'ShowHide Device'. Below the tabs are navigation buttons: 'Addition Doubler', 'Stax', 'Looper', 'Stack Test', and 'Hello World'. Underneath are links for 'Code Gen Save String', 'Interactive', 'Binary2 Decimal', and 'Decimal2 Binary'. A 'Debug' button is also present.

The assembly code window contains the following code:

```
# Store 'Hello world!' at the top of the stack
    .text
    .globl _start
_start:
    li $t0, 0x484f4c4f        # 'Hello'
    li $t1, 0x6f6f6f6f        # ', '
    li $t2, 0x65656565        # ' '
    li $t3, 0x6e6e6e6e        # '!'
    li $t4, 0x0                # null
    li $t5, 0x40               # print string
    li $t6, 0x0                # point to the log

    addi $t0, $t0, 72 # N
    addi $t1, $t1, 41 # e
    addi $t2, $t2, 33 # o
    addi $t3, $t3, 32 # (space)
    addi $t4, $t4, 0
    addi $t5, $t5, 41 # r
    addi $t6, $t6, 1

    addi $t0, $t0, 108 # l
    addi $t1, $t1, 104 # d
    addi $t2, $t2, 105 # i
    addi $t3, $t3, 106 # !
    addi $t4, $t4, 0 # null
    addi $t5, $t5, 108 # t
    addi $t6, $t6, 109 # n

    addi $t0, $t0, 108 # m
    addi $t1, $t1, 105 # e
    addi $t2, $t2, 101 # s
    addi $t3, $t3, 101 # !
    addi $t4, $t4, 0
    addi $t5, $t5, 108 # !
    addi $t6, $t6, 108 # !
```

The registers table shows the following values:

Step	Run	✓ Enable auto switching			
S	T	A	V	Stack	Log
s0	10				
s1	9				
s2	9				
s3	22				
s4	695				
s5	976				
s6	977				
s7	419				

The memory dump table shows the following data:

Address	Value	Label
0x40000000	48 4f 4c 4f 6f 6f 6f 6f 65 65 65 65 6e 6e 6e 6e 00 00 00 00	hello
0x40000008	40 00 00 00	print_string
0x40000010	00 00 00 00	log

(Demo with WeMIPS)

# MIPS Commands

The screenshot shows a MIPS assembly debugger interface. At the top, there are tabs for User, Show/Hide Demo, Addition Counter, Stats, Looper, Stack Test, Hello World, Code Gen Save String, Interactive, Binary/Decimal, Decimal/Binary, and Debug. Below the tabs, there is a code editor window containing the following assembly code:

```
1 # Shows "Hello world!" at the top of the stack
2 .text
3 .globl _start
4 _start:
5    addi   $t0, $zero, 101 # a
6    addi   $t0, $zero, 101 # b
7    addi   $t0, $zero, 100 # 1
8    addi   $t0, $zero, 100 # 1
9    addi   $t0, $zero, 100 # 1
10   addi   $t0, $zero, 100 # 1
11   addi   $t0, $zero, 100 # 1
12   addi   $t0, $zero, 100 # 1
13   addi   $t0, $zero, 100 # 1
14   addi   $t0, $zero, 100 # 1
15   addi   $t0, $zero, 100 # 1
16   addi   $t0, $zero, 100 # 1
17   addi   $t0, $zero, 100 # 1
18   addi   $t0, $zero, 100 # 1
19   addi   $t0, $zero, 100 # 1
20   addi   $t0, $zero, 100 # 1
21   addi   $t0, $zero, 100 # 1
22   addi   $t0, $zero, 100 # d
23   addi   $t0, $zero, 100 # d
24   addi   $t0, $zero, 100 # d
25   addi   $t0, $zero, 100 # d
26   addi   $t0, $zero, 100 # d
27   addi   $t0, $zero, 0 # (null)
28   addi   $t0, $zero, 0 # (null)
29   addi   $t0, $zero, 0 # (null)
30   addi   $t0, $zero, 0 # (null)
31   addi   $t0, $zero, 0 # (null)
32   syscall
```

To the right of the code editor is a register table titled "Registers". It has columns for S, T, A, V, Stack, and Log. The values for each register are as follows:

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

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

# MIPS Commands

The screenshot shows the ShowFide Demo application window. At the top, there are tabs for User, ShowFide Demo, and other developer tools. Below the tabs are buttons for Addition, Counter, IfElse, Looper, StackTest, and Hello World. Further down are buttons for CodeGen, SaveString, Interactive, Binary, Decimal, and Decimal/Binary. A Debug tab is also present. The main area contains assembly code for a "Hello World" program. To the right is a register dump table with columns for S, T, A, V, Stack, and Log.

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

```
1 # Shows "Hello world" at the top of the stack
2 .data
3 .text
4 addi $t0, $zero, 10 # $t0 = 10
5 addi $t0, $zero, 101 # $t0 = 101
6 addi $t0, $zero, 100 # $t0 = 100
7 addi $t0, $zero, 1000 # $t0 = 1000
8 addi $t0, $zero, 100 # $t0 = 100
9 addi $t0, $zero, 1000 # $t0 = 1000
10 addi $t0, $zero, 100 # $t0 = 100
11 addi $t0, $zero, 1000 # $t0 = 1000
12 addi $t0, $zero, 100 # $t0 = 100
13 addi $t0, $zero, 1000 # $t0 = 1000
14 addi $t0, $zero, 100 # $t0 = 100
15 addi $t0, $zero, 1000 # $t0 = 1000
16 addi $t0, $zero, 100 # $t0 = 100
17 addi $t0, $zero, 1000 # $t0 = 1000
18 addi $t0, $zero, 100 # $t0 = 100
19 addi $t0, $zero, 1000 # $t0 = 1000
20 addi $t0, $zero, 100 # $t0 = 100
21 addi $t0, $zero, 1000 # $t0 = 1000
22 addi $t0, $zero, 100 # $t0 = 100
23 addi $t0, $zero, 1000 # $t0 = 1000
24 addi $t0, $zero, 100 # $t0 = 100
25 addi $t0, $zero, 1000 # $t0 = 1000
26 addi $t0, $zero, 100 # $t0 = 100
27 addi $t0, $zero, 0 # (null)
28 addi $t0, $zero, 1000 # $t0 = 1000
29 addi $t0, $zero, 100 # $t0 = 100
30 addi $t0, $zero, 1000 # $t0 = 1000
31 addi $t0, $zero, 100 # $t0 = 100
32 addi $t0, $zero, 1000 # $t0 = 1000
33 syscall
```

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

# MIPS Commands

The screenshot shows a software interface for assembly code. At the top, there's a menu bar with 'File', 'Edit', 'Run', 'Help', and tabs for 'ShowCode Demo', 'Addition Counter', 'Stack Test', 'Loop', 'Stack Test -', 'Hello World', 'Code Gen Save String', 'Interactive', 'Binary Decimal', 'Decimal Binary', and 'Debug'. Below the tabs is a toolbar with icons for 'Run', 'Break', 'Stop', and 'Reset'. The main area contains assembly code and a register dump.

**Assembly Code:**

```
1 # Shows "Hello world!" at the top of the stack
2 .text
3 .globl _start
4 _start:
5    addiu   $sp,$sp,-16      # allocate space for stack frame
6    addiu   $t0,$zero,100      # set base pointer to stack
7    addiu   $t0,$t0,100      # set stack pointer to base + 100
8    addiu   $t0,$t0,100      # set stack pointer to base + 200
9    addiu   $t0,$t0,100      # set stack pointer to base + 300
10   addiu   $t0,$t0,100      # set stack pointer to base + 400
11   addiu   $t0,$t0,100      # set stack pointer to base + 500
12   addiu   $t0,$t0,100      # set stack pointer to base + 600
13   addiu   $t0,$t0,100      # set stack pointer to base + 700
14   addiu   $t0,$t0,100      # set stack pointer to base + 800
15   addiu   $t0,$t0,100      # set stack pointer to base + 900
16   addiu   $t0,$t0,100      # set stack pointer to base + 1000
17   addiu   $t0,$t0,100      # set stack pointer to base + 1100
18   addiu   $t0,$t0,100      # set stack pointer to base + 1200
19   addiu   $t0,$t0,100      # set stack pointer to base + 1300
20   addiu   $t0,$t0,100      # set stack pointer to base + 1400
21   addiu   $t0,$t0,100      # set stack pointer to base + 1500
22   addiu   $t0,$t0,100      # set stack pointer to base + 1600
23   addiu   $t0,$t0,100      # set stack pointer to base + 1700
24   addiu   $t0,$t0,100      # set stack pointer to base + 1800
25   addiu   $t0,$t0,100      # set stack pointer to base + 1900
26   addiu   $t0,$t0,100      # set stack pointer to base + 2000
27   addiu   $t0,$t0,100      # set stack pointer to base + 2100
28   addiu   $t0,$t0,100      # set stack pointer to base + 2200
29   addiu   $t0,$t0,100      # set stack pointer to base + 2300
30   addiu   $t0,$t0,100      # set stack pointer to base + 2400
31   addiu   $t0,$t0,100      # set stack pointer to base + 2500
32   addiu   $t0,$t0,100      # set stack pointer to base + 2600      # print to the log
33   syscall
```

**Register Dump:**

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

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

# MIPS Commands

The screenshot shows the ShowFide Demo software interface. At the top, there's a menu bar with 'File', 'Edit', 'Run', 'User Guide', 'Unit Tests', and 'Doc'. Below the menu is a toolbar with buttons for 'Addition Counter', 'Btav', 'Looper', 'Stack Test', 'Hello World', 'Code Gen Save String', 'Interactive', 'Binary Decimal', 'Decimal Binary', and 'Debug'. The main window contains two panes. The left pane displays assembly code:

```
1 # Shows "Hello world!" at the top of the stack
2 .text
3 .globl _start
4 _start:
5    addi   $s0,$zero,100 # s
6    addi   $s1,$zero,100 # t
7    addi   $t0,$zero,100 # v
8    addi   $s0,$zero,100 # s
9    addi   $s1,$zero,100 # t
10   addi   $t0,$zero,100 # v
11   addi   $s0,$zero,100 # s
12   addi   $s1,$zero,100 # t
13   addi   $t0,$zero,100 # v
14   addi   $s0,$zero,100 # s
15   addi   $s1,$zero,100 # t
16   addi   $t0,$zero,100 # v
17   addi   $s0,$zero,100 # s
18   addi   $s1,$zero,100 # t
19   addi   $t0,$zero,100 # v
20   addi   $s0,$zero,100 # s
21   addi   $s1,$zero,100 # t
22   addi   $t0,$zero,100 # v
23   addi   $s0,$zero,100 # s
24   addi   $s1,$zero,100 # t
25   addi   $t0,$zero,100 # v
26   addi   $s0,$zero,100 # s
27   addi   $s1,$zero,100 # t
28   addi   $t0,$zero,100 # v
29   addi   $s0,$zero,100 # s
30   addi   $s1,$zero,100 # t
31   addi   $t0,$zero,100 # v
32   addi   $s0,$zero,4 # s is for print string
33   addi   $s0,$zero,5 # print to the log
34   syscall
```

The right pane shows a register dump table with columns for S, T, A, V, Stack, and Log. The registers are numbered \$0 to \$T:

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

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

# MIPS Commands

The screenshot shows the ShowMIPS Demo application window. At the top, there are tabs for User, ShowMIPS Demo, and Run. Below the tabs are several menu items: Addition, Counter, Iters, Looper, Stack Test, Hello World, Code Gen, Save String, Interactive, Binary, Decimal, and Decimal/Binary. A Debug tab is also present. On the right side of the window, there is a "User Guide | Unit Tests | Docs" link. The main area contains assembly code and a register dump.

```
# Shows "Hello world" at the top of the stack
1    .text
2    .globl _start
3    .type _start, @function
4    _start:
5        addiu   $sp,$sp,-16      # allocate space for arguments
6        addiu   $sp,$sp,-16      # allocate space for local variables
7        addiu   $sp,$sp,-16      # allocate space for local variables
8        addiu   $sp,$sp,-16      # allocate space for local variables
9        addiu   $sp,$sp,-16      # allocate space for local variables
10       addiu   $sp,$sp,-16      # allocate space for local variables
11       addiu   $sp,$sp,-16      # allocate space for local variables
12       addiu   $sp,$sp,-16      # allocate space for local variables
13       addiu   $sp,$sp,-16      # allocate space for local variables
14       addiu   $sp,$sp,-16      # allocate space for local variables
15       addiu   $sp,$sp,-16      # allocate space for local variables
16       addiu   $sp,$sp,-16      # allocate space for local variables
17       addiu   $sp,$sp,-16      # allocate space for local variables
18       addiu   $sp,$sp,-16      # allocate space for local variables
19       addiu   $sp,$sp,-16      # allocate space for local variables
20       addiu   $sp,$sp,-16      # allocate space for local variables
21       addiu   $sp,$sp,-16      # allocate space for local variables
22       addiu   $sp,$sp,-16      # allocate space for local variables
23       addiu   $sp,$sp,-16      # allocate space for local variables
24       addiu   $sp,$sp,-16      # allocate space for local variables
25       addiu   $sp,$sp,-16      # allocate space for local variables
26       addiu   $sp,$sp,-16      # allocate space for local variables
27       addiu   $sp,$sp,-16      # allocate space for local variables
28       addiu   $sp,$sp,-16      # allocate space for local variables
29       addiu   $sp,$sp,-16      # allocate space for local variables
30       addiu   $sp,$sp,-16      # allocate space for local variables
31       addiu   $sp,$sp,-16      # allocate space for local variables
32       addiu   $sp,$sp,-16      # allocate space for local variables
33       addiu   $sp,$sp,-16      # print to the log
34       syscall
```

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

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

# MIPS Commands

The screenshot shows a software interface for generating MIPS assembly code. At the top, there are tabs for "ShowCode Demo", "User Guide", "Unit Tests", and "Documentation". Below the tabs are buttons for "Addition", "Calculator", "Itave", "Looper", "Stack Test", "Hello World", "Code Gen Save String", "Interactive", "Binary", "Decimal", "Hexadecimal", and "Debug".

The main area displays the following MIPS assembly code:

```
# Shows "Hello world" at the top of the stack
1    .data
2        msg: .asciiz "Hello world"
3
4    .text
5        la $t0, msg
6        li $t1, 10
7        add $t2, $t0, $t1 # 10
8        add $t2, $t2, $t2 # 10*2 = 20
9        add $t2, $t2, $t2 # 10*2*2 = 40
10       add $t2, $t2, $t2 # 10*2*2*2 = 80
11       add $t2, $t2, $t2 # 10*2*2*2*2 = 160
12       add $t2, $t2, $t2 # 10*2*2*2*2*2 = 320
13       add $t2, $t2, $t2 # 10*2*2*2*2*2*2 = 640
14       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2 = 1280
15       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2 = 2560
16       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2 = 5120
17       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2 = 10240
18       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2 = 20480
19       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2*2 = 40960
20       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2*2*2 = 81920
21       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2*2*2*2 = 163840
22       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2 = 327680
23       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2 = 655360
24       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2 = 1310720
25       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2 = 2621440
26       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2 = 5242880
27       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2 = 10485760
28       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2 = 20971520
29       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2 = 41943040
30       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2 = 83886080
31       add $t2, $t2, $t2 # 10*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2 = 167772160
32       add $t2, $t2, $t2 # print to the log
            syscall
```

To the right of the assembly code is a register dump table with columns for \$, T, A, V, Stack, and Log. The registers shown are \$0 through \$31, with their values listed below:

\$	T	A	V	Stack	Log
\$0				10	
\$1				9	
\$2				8	
\$3				7	
\$4				6	
\$5				5	
\$6				4	
\$7				3	
\$8				2	
\$9				1	
\$10				0	
\$11				41943040	
\$12				83886080	
\$13				167772160	
\$14				335544320	
\$15				671088640	
\$16				1342177280	
\$17				2684354560	
\$18				5368709120	
\$19				10737418240	
\$20				21474836480	
\$21				42949673600	
\$22				85899347200	
\$23				171798694400	
\$24				343597388800	
\$25				687194777600	
\$26				1374389555200	
\$27				2748779110400	
\$28				5497558220800	
\$29				10995116441600	
\$30				21990232883200	
\$31				43980465766400	

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

# MIPS Commands

The screenshot shows a MIPS assembly debugger interface. At the top, there are tabs for "Show/Hide Demo", "User Guide", and "Unit Tests". Below the tabs are several menu options: Addition, Subtraction, Bitwise, Looper, Stack Test, Hello World, Code Gen, Save String, Interactive, Binary/Decimal, Decimal/Binary, and Debug. The "Debug" tab is currently selected.

The main area displays the assembly code for "Hello World" and its corresponding memory dump. The assembly code is:

```
# Shows "Hello world" at the top of the stack
1    .data
2    msg: .asciiz "Hello world\n"
3
4    .text
5    .globl _start
6    _start:
7        addi $t0, $zero, 100 # $t0 = 100
8        addi $t1, $zero, 100 # $t1 = 100
9        addi $t2, $zero, 100 # $t2 = 100
10       addi $t3, $zero, 100 # $t3 = 100
11       addi $t4, $zero, 100 # $t4 = 100
12       addi $t5, $zero, 100 # $t5 = 100
13       addi $t6, $zero, 100 # $t6 = 100
14       addi $t7, $zero, 100 # $t7 = 100
15       addi $t8, $zero, 100 # $t8 = 100
16       addi $t9, $zero, 100 # $t9 = 100
17       addi $t10, $zero, 100 # $t10 = 100
18       addi $t11, $zero, 100 # $t11 = 100
19       addi $t12, $zero, 100 # $t12 = 100
20       addi $t13, $zero, 100 # $t13 = 100
21       addi $t14, $zero, 100 # $t14 = 100
22       addi $t15, $zero, 100 # $t15 = 100
23       addi $t16, $zero, 100 # $t16 = 100
24       addi $t17, $zero, 100 # $t17 = 100
25       addi $t18, $zero, 100 # $t18 = 100
26       addi $t19, $zero, 100 # $t19 = 100
27       addi $t20, $zero, 100 # $t20 = 100
28       addi $t21, $zero, 100 # $t21 = 100
29       addi $t22, $zero, 100 # $t22 = 100
30       addi $t23, $zero, 100 # $t23 = 100
31       addi $t24, $zero, 100 # $t24 = 100
32       addi $t25, $zero, 100 # $t25 = 100
33       addi $t26, $zero, 100 # $t26 = 100
34       addi $t27, $zero, 100 # $t27 = 100
35       addi $t28, $zero, 100 # $t28 = 100
36       addi $t29, $zero, 100 # $t29 = 100
37       addi $t30, $zero, 100 # $t30 = 100
38       addi $t31, $zero, 100 # $t31 = 100
39
40       addi $t0, $zero, 4 # $t0 = 4 for print string
41       li $v0, 4
42       move $a0, $t0
43       syscall
44
45       li $v0, 10
46       syscall
```

Below the assembly code is a memory dump table with columns for S, T, A, V, Stack, and Log. The table shows the values of registers \$0 through \$31 across four rows. The values are:

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

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **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 assembly debugger interface. At the top, there are tabs for 'Show/Hide Demo' (selected), 'User Guide', 'Unit Tests', and 'Docs'. Below the tabs are several menu options: Addition, Subtraction, Bitwise, Looper, Stack Test, and Hello World. Underneath these are sub-options: Code Gen, Save String, Interactive, Binary, Decimal, and Hexadecimal. A 'Debug' tab is also present.

The main area displays assembly code:

```
# Shows "Hello world" at the top of the stack
1    .data
2    msg: .asciiz "Hello world\n"
3    .text
4    la $t0, msg
5    addi $t0, $t0, 10 # a
6    addi $t0, $t0, 10 # b
7    addi $t0, $t0, 10 # c
8    addi $t0, $t0, 10 # d
9    addi $t0, $t0, 10 # e
10   addi $t0, $t0, 10 # f
11   addi $t0, $t0, 10 # g
12   addi $t0, $t0, 10 # h
13   addi $t0, $t0, 10 # i
14   addi $t0, $t0, 10 # j
15   addi $t0, $t0, 10 # k
16   addi $t0, $t0, 10 # l
17   addi $t0, $t0, 10 # m
18   addi $t0, $t0, 10 # n
19   addi $t0, $t0, 10 # o
20   addi $t0, $t0, 10 # p
21   addi $t0, $t0, 10 # q
22   addi $t0, $t0, 10 # r
23   addi $t0, $t0, 10 # s
24   addi $t0, $t0, 10 # t
25   addi $t0, $t0, 10 # u
26   addi $t0, $t0, 10 # v
27   addi $t0, $t0, 10 # w
28   addi $t0, $t0, 10 # x
29   addi $t0, $t0, 10 # y
30   addi $t0, $t0, 10 # z
31   addi $t0, $t0, 10 # {newline}
32   li $v0, 4 # $t0 = 10
33   syscall
34           # print to the log
```

To the right of the assembly code, there is a register table titled 'Registers'. It has columns for \$, T, A, V, Stack, and Log. The values for each register are:

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

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **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 assembly debugger interface. The code window displays the assembly code for a "Hello World" program, starting with a stack initialization and followed by standard library calls for printing strings. The register window shows the current values for \$s0 through \$t1, \$v0, \$a0, and \$a1.

```
# Show "Hello world" at the top of the stack
1    .data
2    .text
3    .globl _start
4    _start:
5    addi   $sp,$sp,-16      # a
6    addi   $t0,$zero,100      # b
7    addi   $t0,$t0,1           # c
8    addi   $t0,$t0,1           # d
9    addi   $t0,$t0,1           # e
10   addi   $t0,$t0,1           # f
11   addi   $t0,$t0,1           # g
12   addi   $t0,$t0,1           # h
13   addi   $t0,$t0,1           # i
14   addi   $t0,$t0,1           # j
15   addi   $t0,$t0,1           # k
16   addi   $t0,$t0,1           # l
17   addi   $t0,$t0,1           # m
18   addi   $t0,$t0,1           # n
19   addi   $t0,$t0,1           # o
20   addi   $t0,$t0,1           # p
21   addi   $t0,$t0,1           # q
22   addi   $t0,$t0,1           # r
23   addi   $t0,$t0,1           # s
24   addi   $t0,$t0,1           # t
25   addi   $t0,$t0,1           # u
26   addi   $t0,$t0,1           # v
27   addi   $t0,$t0,1           # w
28   addi   $t0,$t0,1           # x
29   addi   $t0,$t0,1           # y
30   addi   $t0,$t0,1           # z
31   addi   $t0,$t0,4           # for print string
32   li    $a0,$msg             # print to the log
33   syscall
```

S	T	A	V	Stack	Log
\$0	13				
\$1	9				
\$2	22				
\$3	60				
\$4	61				
\$5	807				
\$6	418				
\$7					

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

# Challenge:

Line: 3 Go! Show/Hide Demos

User Guide | Unit Tests | Docs

Addition Doubler Stav Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

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

Step Run  Enable auto switching

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

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

# WeMIPS

User | 3 | Dat ShowHide Device

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

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

Debug

```
# Store 'Hello world!' at the top of the stack
1    ADDI $t0, $zero, 72 # $N
2    ADDI $t1, $zero, 101 # $e
3    ADDI $t2, $zero, 101 # $n
4    ADDI $t3, $zero, 104 # $l
5    ADDI $t4, $zero, 108 # $o
6    ADDI $t5, $zero, 108 # $w
7    ADDI $t6, $zero, 108 # $r
8    ADDI $t7, $zero, 108 # $d
9    ADDI $t8, $zero, 108 # $c
10   ADDI $t9, $zero, 108 # $o
11   ADDI $t10, $zero, 108 # $n
12   ADDI $t11, $zero, 108 # $d (space)
13   ADDI $t12, $zero, 108 # $e
14   ADDI $t13, $zero, 108 # $n
15   ADDI $t14, $zero, 108 # $l
16   ADDI $t15, $zero, 108 # $o
17   ADDI $t16, $zero, 108 # $w
18   ADDI $t17, $zero, 108 # $r
19   ADDI $t18, $zero, 108 # $d
20   ADDI $t19, $zero, 108 # $c
21   ADDI $t20, $zero, 108 # $o
22   ADDI $t21, $zero, 108 # $n
23   ADDI $t22, $zero, 108 # $d
24   ADDI $t23, $zero, 108 # $c
25   ADDI $t24, $zero, 33 # $!
26   ADDI $t25, $zero, 110 # $l
27   ADDI $t26, $zero, 9 # $(null)
28   ADDI $t27, $zero, 110 # $l
29   ADDI $t28, $zero, 110 # $l
30   ADDI $t29, $zero, 4 # 4 is for print string
31   ADDI $t30, $zero, 0      # point to the log
32   syscall
```

Step	Run	<input checked="" type="checkbox"/> Enable auto switching			
S	T	A	V	Stack	Log
s0:	10				
s1:	9				
s2:	8				
s3:	22				
s4:	696				
s5:	976				
s6:	977				
s7:	419				

(Demo with WeMIPS)

# Today's Topics



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

# Loops & Jumps in Machine Language

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



A screenshot of a debugger interface. On the left, the assembly code window shows a series of memory addresses followed by assembly instructions, mostly consisting of `MOV R1, R2` or `MOV R1, R2, R3`. On the right, the registers window displays various CPU registers with their current 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.



A screenshot of a hex editor application. The left pane shows a list of memory pages, each containing a series of memory addresses and their corresponding byte values. The right pane is a detailed view of a specific page, showing memory addresses from 0x00000000 to 0x0000000F. The bytes displayed are: 48 45 4C 4C 4D 4E 4F 4F 4A 4B 4C 4D 4E 4F 4F 4A 4B 4C. Below the address 0x00000000, there is a label "Label1". Above the first byte (48), there is a jump instruction (opcode C7 00). The bottom of the window has standard file menu options like File, Edit, View, Insert, Search, and Help.

# Loops & Jumps in Machine Language

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



# Loops & Jumps in Machine Language

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



# Loops & Jumps in Machine Language

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



# Loops & Jumps in Machine Language

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



# Loops & Jumps in Machine Language

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



# Jump Demo

Line: 18 Go!

Show/Hide Demos

User Guide | Unit Tests | Docs

```
1 ADDI $sp, $sp, -27      # Set up stack
2 ADDI $s3, $zero, 1       # Store 1 in a register
3 ADDI $t0, $zero, 97      # Set $t0 at 97 (a)
4 ADDI $s2, $zero, 26      # Use to test when you reach 26
5 SETUP: SB $t0, 0($sp)    # Next letter in $t0
6 ADDI $sp, $sp, 1         # Increment the stack
7 SUB $s2, $s2, $s3        # Decrease the counter by 1
8 ADDI $t0, $t0, 1         # Increment the letter
9 BEQ $s2, $zero, DONE     # Jump to done if $s2 == 0
10 J SETUP
11 J SETUP
12 DONE: ADDI $t0, $zero, 0 # Null (0) to terminate string
13 SB $t0, 0($sp)          # Add null to stack
14 ADDI $sp, $sp, -26      # Set up stack to print
15 ADDI $v0, $zero, 4       # 4 is for print string
16 ADDI $a0, $sp, 0         # Set $a0 to stack pointer
17 syscall                # Print to the log
```

(Demo  
with  
WeMIPS)

Step Run  Enable auto switching

S T A V Stack Log

Clear Log

Emulation complete, returning to line 1

abcdefghijklmnopqrstuvwxyz

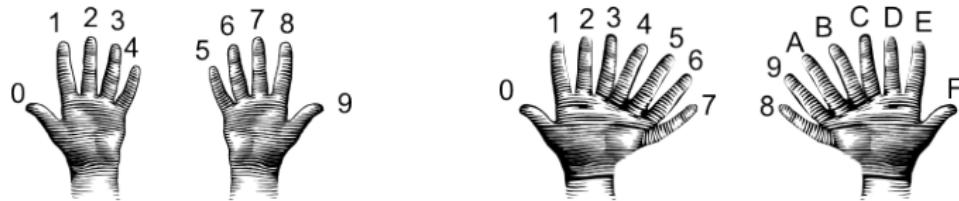


# Today's Topics



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

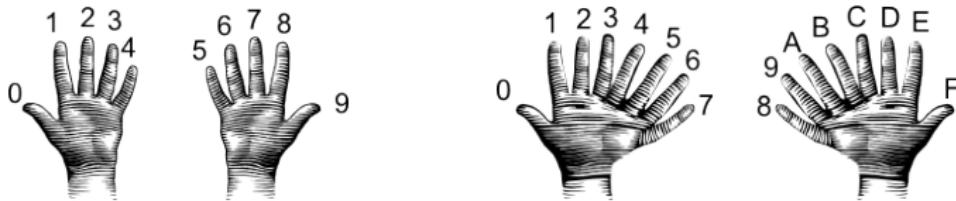
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

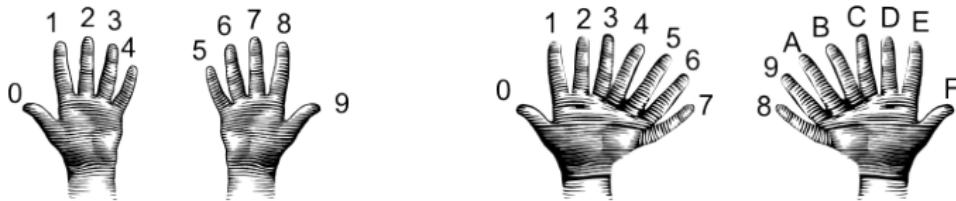
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

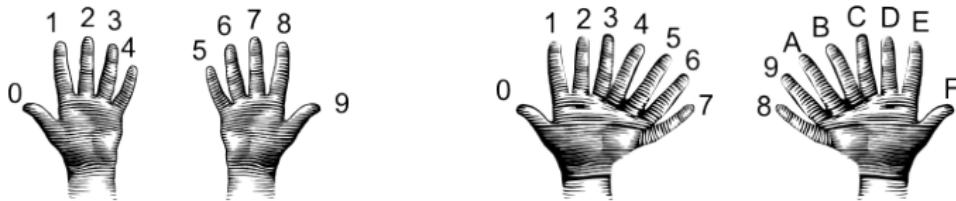
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?

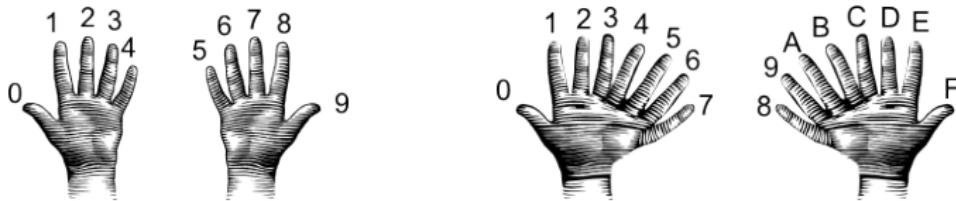
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?  
2 in decimal is 2.

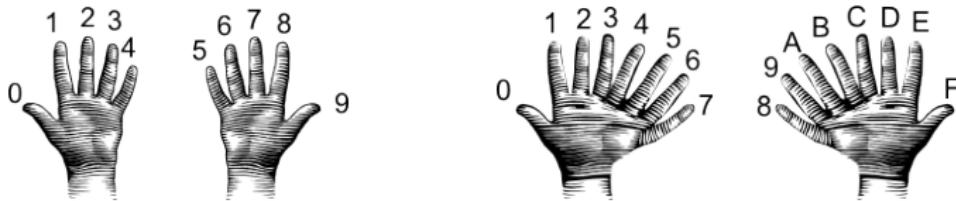
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?  
2 in decimal is 2.  $2 \times 16$  is 32.

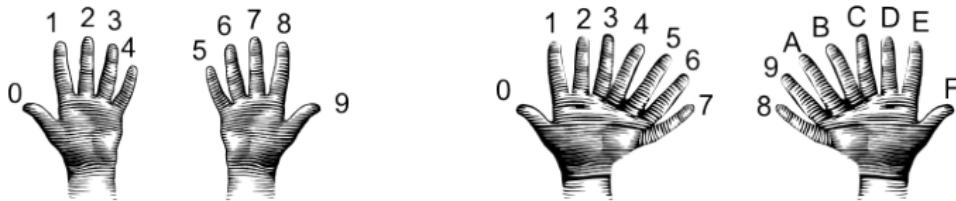
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?  
2 in decimal is 2.  $2 \times 16$  is 32.  
A in decimal digits is 10.

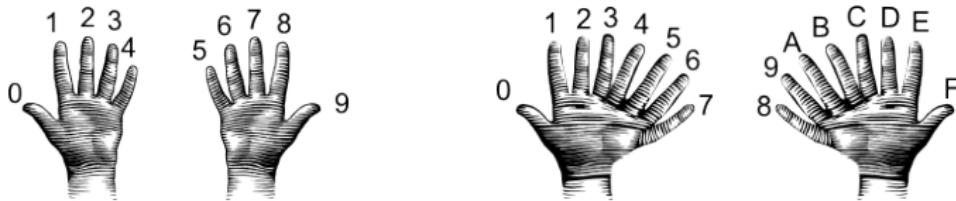
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?  
2 in decimal is 2.  $2 \times 16$  is 32.  
A in decimal digits is 10.  
 $32 + 10$  is 42.

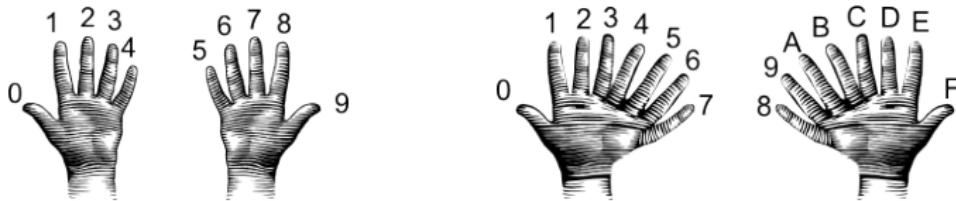
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?  
2 in decimal is 2.  $2 \times 16$  is 32.  
A in decimal digits is 10.  
 $32 + 10$  is 42.  
Answer is 42.
  - Example: what is 99 as a decimal number?

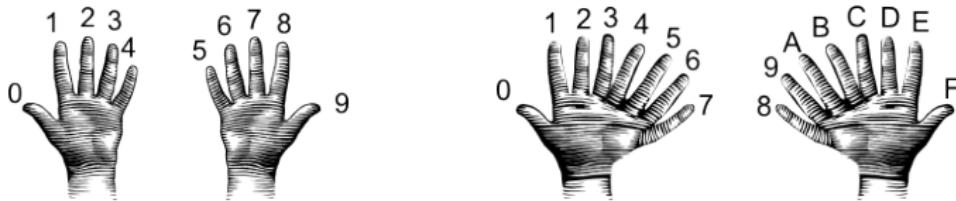
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?  
2 in decimal is 2.  $2 \times 16$  is 32.  
A in decimal digits is 10.  
 $32 + 10$  is 42.  
Answer is 42.
  - Example: what is 99 as a decimal number?  
9 in decimal is 9.

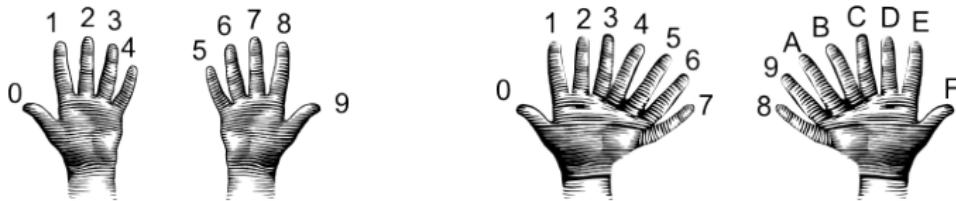
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?  
2 in decimal is 2.  $2 \times 16$  is 32.  
A in decimal digits is 10.  
 $32 + 10$  is 42.  
Answer is 42.
  - Example: what is 99 as a decimal number?  
9 in decimal is 9.  $9 \times 16$  is 144.

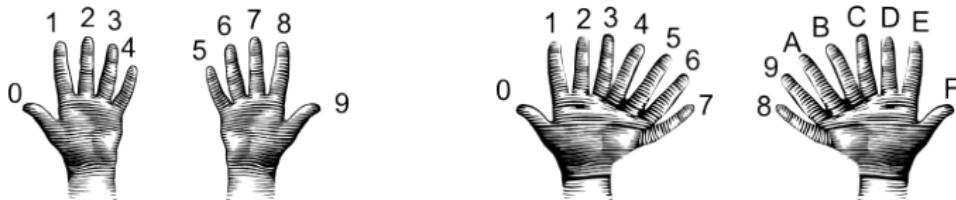
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?  
2 in decimal is 2.  $2 \times 16$  is 32.  
A in decimal digits is 10.  
 $32 + 10$  is 42.  
Answer is 42.
  - Example: what is 99 as a decimal number?  
9 in decimal is 9.  $9 \times 16$  is 144.  
9 in decimal digits is 9

# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

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

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

A in decimal digits is 10.

$32 + 10$  is 42.

Answer is 42.

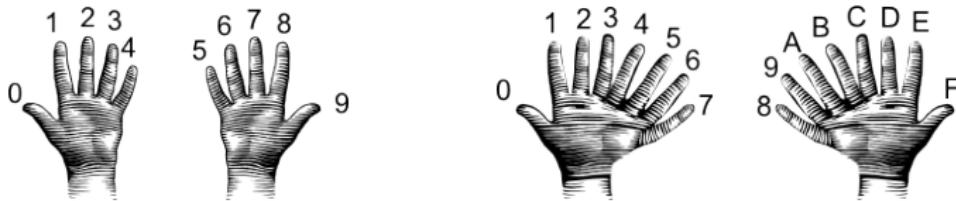
- Example: what is 99 as a decimal number?

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

9 in decimal digits is 9

$144 + 9$  is 153.

# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

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

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

A in decimal digits is 10.

$32 + 10$  is 42.

Answer is 42.

- Example: what is 99 as a decimal number?

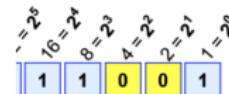
9 in decimal is 9.  $9 \times 16$  is 144.

9 in decimal digits is 9

$144 + 9$  is 153.

Answer is 153.

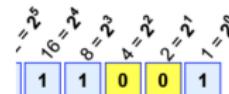
# Decimal to Binary: Converting Between Bases



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

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

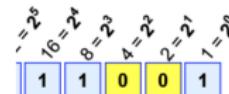
# Decimal to Binary: Converting Between Bases



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

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

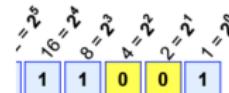
# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:
  - Divide by  $128 (= 2^7)$ . Quotient is the first digit.
  - Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
  - Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.

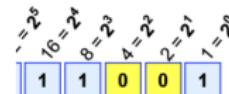
# Decimal to Binary: Converting Between Bases



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

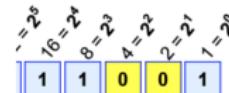
- From decimal to binary:
  - ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
  - ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
  - ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
  - ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.

# Decimal to Binary: Converting Between Bases



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

# Decimal to Binary: Converting Between Bases

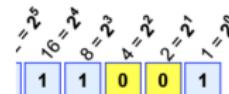


Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

- From decimal to binary:

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

# Decimal to Binary: Converting Between Bases

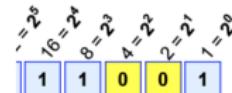


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From decimal to binary:

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

# Decimal to Binary: Converting Between Bases

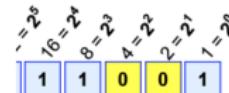


Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 1 = 25$

- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.

# Decimal to Binary: Converting Between Bases

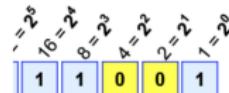


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

# Decimal to Binary: Converting Between Bases



Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 0 \times 1 = 16 + 8 + 4 = 28$

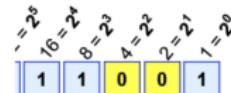
- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.

- Example: what is 130 in binary notation?

130/128 is 1 rem 2.

# Decimal to Binary: Converting Between Bases



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

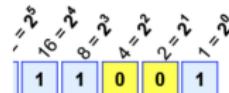
- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.

- Example: what is 130 in binary notation?

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

# Decimal to Binary: Converting Between Bases



$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From decimal to binary:

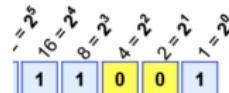
- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.

- Example: what is 130 in binary notation?

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

2/64 is 0 rem 2.

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

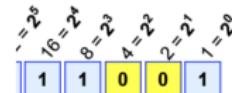
- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.

Example: what is 130 in binary notation?

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

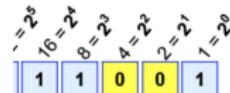
- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.

- Example: what is 130 in binary notation?

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

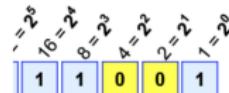
- Divide by 128 ( $= 2^7$ ). Quotient is the first digit.
- Divide remainder by 64 ( $= 2^6$ ). Quotient is the next digit.
- Divide remainder by 32 ( $= 2^5$ ). Quotient is the next digit.
- Divide remainder by 16 ( $= 2^4$ ). Quotient is the next digit.
- Divide remainder by 8 ( $= 2^3$ ). Quotient is the next digit.
- Divide remainder by 4 ( $= 2^2$ ). Quotient is the next digit.
- Divide remainder by 2 ( $= 2^1$ ). Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

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

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

2/32 is 0 rem 2.

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

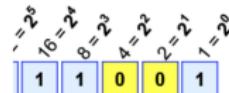
- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

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

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

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

# Decimal to Binary: Converting Between Bases



Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

- From decimal to binary:

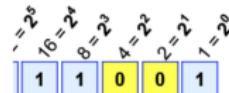
- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

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

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

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

# Decimal to Binary: Converting Between Bases



Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

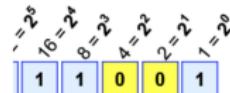
130/128 is 1 rem 2. First digit is 1: 1...

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

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

2/16 is 0 rem 2.

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

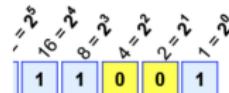
130/128 is 1 rem 2. First digit is 1: 1...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.

- Example: what is 130 in binary notation?

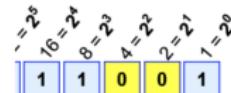
130/128 is 1 rem 2. First digit is 1: 1...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

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

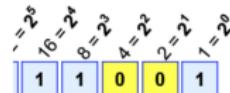
2/64 is 0 rem 2. Next digit is 0: 10...

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

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

2/8 is 0 rem 2.

# Decimal to Binary: Converting Between Bases



Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

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

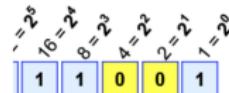
2/64 is 0 rem 2. Next digit is 0: 10...

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

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

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

# Decimal to Binary: Converting Between Bases



Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.

- Example: what is 130 in binary notation?

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

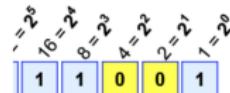
2/64 is 0 rem 2. Next digit is 0: 10...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

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

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

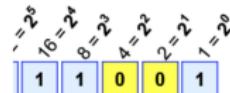
2/32 is 0 rem 2. Next digit is 0: 100...

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

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

2/4 is 0 remainder 2.

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.

- Example: what is 130 in binary notation?

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

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

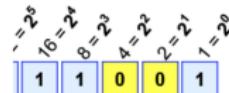
2/32 is 0 rem 2. Next digit is 0: 100...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

- Divide by 128 ( $= 2^7$ ). Quotient is the first digit.
- Divide remainder by 64 ( $= 2^6$ ). Quotient is the next digit.
- Divide remainder by 32 ( $= 2^5$ ). Quotient is the next digit.
- Divide remainder by 16 ( $= 2^4$ ). Quotient is the next digit.
- Divide remainder by 8 ( $= 2^3$ ). Quotient is the next digit.
- Divide remainder by 4 ( $= 2^2$ ). Quotient is the next digit.
- Divide remainder by 2 ( $= 2^1$ ). Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

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

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

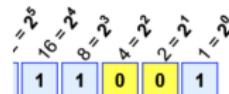
2/32 is 0 rem 2. Next digit is 0: 100...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

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

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

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

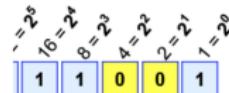
2/16 is 0 rem 2. Next digit is 0: 1000...

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

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

2/2 is 1 rem 0.

# Decimal to Binary: Converting Between Bases



Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 1 \times 1 = 16 + 8 + 4 + 1 = 29$

- From decimal to binary:

- Divide by 128 ( $= 2^7$ ). Quotient is the first digit.
- Divide remainder by 64 ( $= 2^6$ ). Quotient is the next digit.
- Divide remainder by 32 ( $= 2^5$ ). Quotient is the next digit.
- Divide remainder by 16 ( $= 2^4$ ). Quotient is the next digit.
- Divide remainder by 8 ( $= 2^3$ ). Quotient is the next digit.
- Divide remainder by 4 ( $= 2^2$ ). Quotient is the next digit.
- Divide remainder by 2 ( $= 2^1$ ). Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

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

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

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

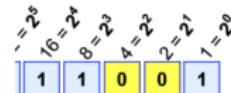
2/16 is 0 rem 2. Next digit is 0: 1000...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

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

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

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

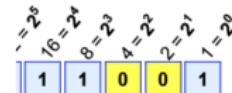
2/16 is 0 rem 2. Next digit is 0: 1000...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

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

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

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

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

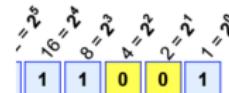
2/8 is 0 rem 2. Next digit is 0: 10000...

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

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

Adding the last remainder: 10000010

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

- Divide by 128 ( $= 2^7$ ). Quotient is the first digit.
- Divide remainder by 64 ( $= 2^6$ ). Quotient is the next digit.
- Divide remainder by 32 ( $= 2^5$ ). Quotient is the next digit.
- Divide remainder by 16 ( $= 2^4$ ). Quotient is the next digit.
- Divide remainder by 8 ( $= 2^3$ ). Quotient is the next digit.
- Divide remainder by 4 ( $= 2^2$ ). Quotient is the next digit.
- Divide remainder by 2 ( $= 2^1$ ). Quotient is the next digit.
- The last remainder is the last digit.

- Example: what is 130 in binary notation?

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

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

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

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

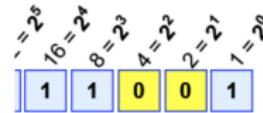
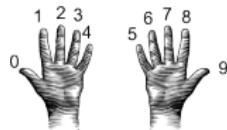
2/8 is 0 rem 2. Next digit is 0: 10000...

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

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

Adding the last remainder: 10000010

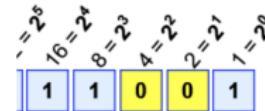
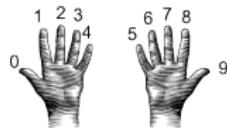
# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

# Decimal to Binary: Converting Between Bases

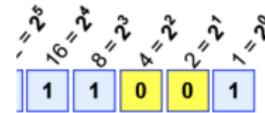
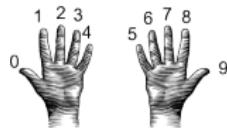


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

$99/128$  is 0 rem 99.

# Decimal to Binary: Converting Between Bases

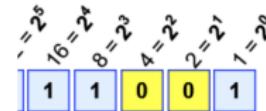
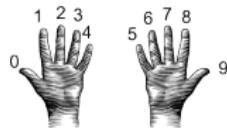


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

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

# Decimal to Binary: Converting Between Bases



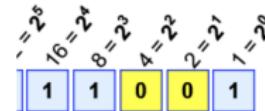
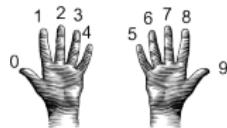
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

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

99/64 is 1 rem 35.

# Decimal to Binary: Converting Between Bases



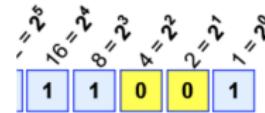
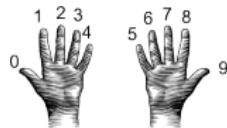
Example:  $1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

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

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

# Decimal to Binary: Converting Between Bases



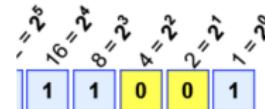
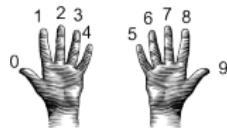
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

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

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

# Decimal to Binary: Converting Between Bases



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

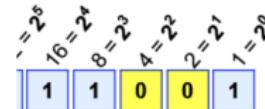
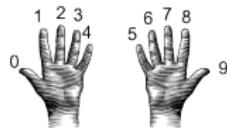
- Example: what is 99 in binary notation?

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

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

35/32 is 1 rem 3.

# Decimal to Binary: Converting Between Bases



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

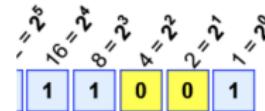
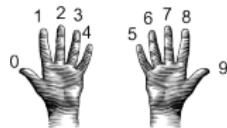
- Example: what is 99 in binary notation?

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

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

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

# Decimal to Binary: Converting Between Bases



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

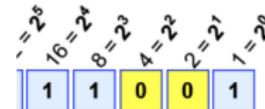
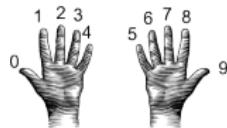
- Example: what is 99 in binary notation?

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

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

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

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

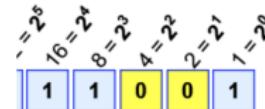
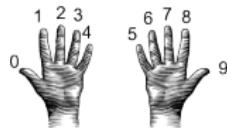
99/128 is 0 rem 99. First digit is 0: 0...

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

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

3/16 is 0 rem 3.

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

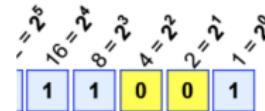
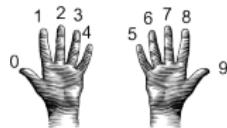
99/128 is 0 rem 99. First digit is 0: 0...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

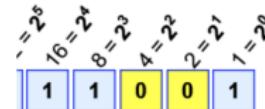
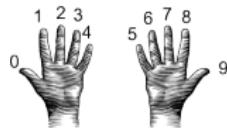
99/128 is 0 rem 99. First digit is 0: 0...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

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

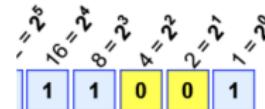
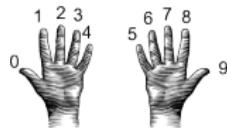
99/64 is 1 rem 35. Next digit is 1: 01...

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

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

3/8 is 0 rem 3.

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

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

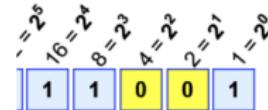
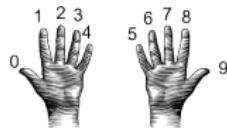
99/64 is 1 rem 35. Next digit is 1: 01...

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

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

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

# Decimal to Binary: Converting Between Bases



Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

- Example: what is 99 in binary notation?

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

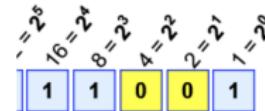
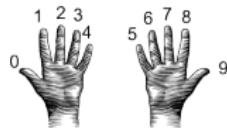
99/64 is 1 rem 35. Next digit is 1: 01...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

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

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

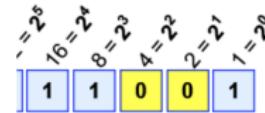
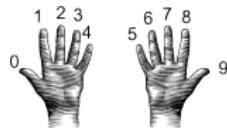
35/32 is 1 rem 3. Next digit is 1: 011...

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

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

3/4 is 0 remainder 3.

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

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

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

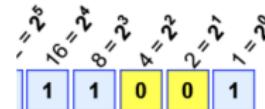
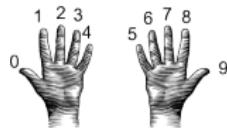
35/32 is 1 rem 3. Next digit is 1: 011...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

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

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

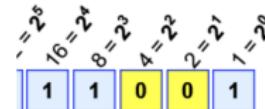
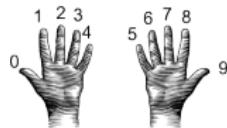
35/32 is 1 rem 3. Next digit is 1: 011...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

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

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

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

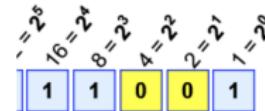
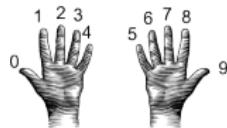
3/16 is 0 rem 3. Next digit is 0: 0110...

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

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

3/2 is 1 rem 1.

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

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

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

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

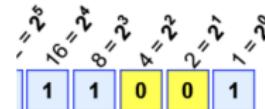
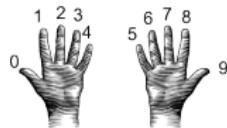
3/16 is 0 rem 3. Next digit is 0: 0110...

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

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

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

# Decimal to Binary: Converting Between Bases

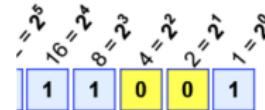
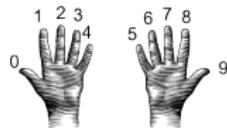


Example:  $1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

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

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

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

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

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

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

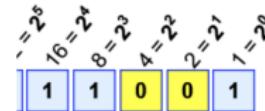
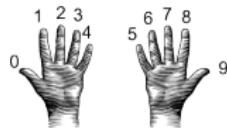
3/8 is 0 rem 3. Next digit is 0: 01100...

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

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

Adding the last remainder: 01100011

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

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

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

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

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

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

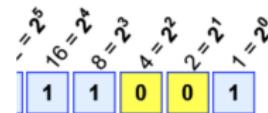
3/4 is 0 remainder 3. Next digit is 0: 011000...

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

Adding the last remainder: 01100011

Answer is 1100011.

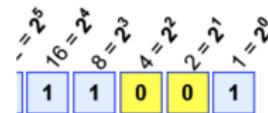
# Binary to Decimal: Converting Between Bases



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

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

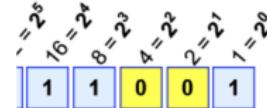
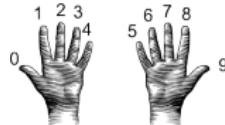
# Binary to Decimal: Converting Between Bases



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

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

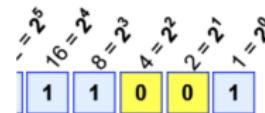
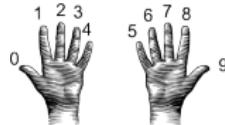
# Binary to Decimal: Converting Between Bases



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

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

# Binary to Decimal: Converting Between Bases

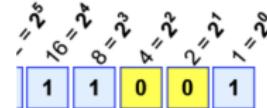


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

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

# Binary to Decimal: Converting Between Bases

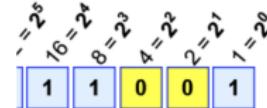
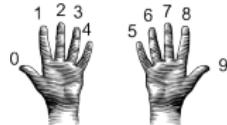


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

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

# Binary to Decimal: Converting Between Bases

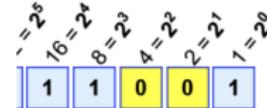


$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

# Binary to Decimal: Converting Between Bases

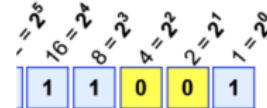
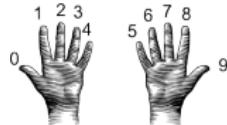


$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

# Binary to Decimal: Converting Between Bases

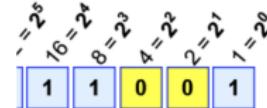


$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

# Binary to Decimal: Converting Between Bases

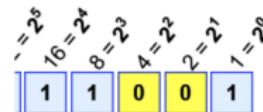
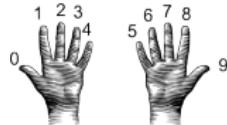


$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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



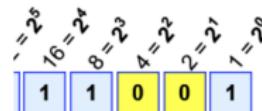
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

Sum starts with:

# Binary to Decimal: Converting Between Bases



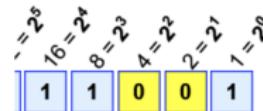
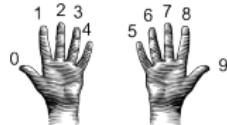
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

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

# Binary to Decimal: Converting Between Bases



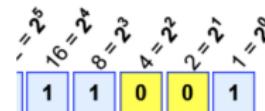
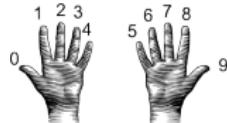
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$$

- From binary to decimal:

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

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

# Binary to Decimal: Converting Between Bases



$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

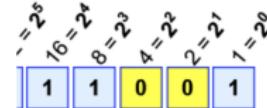
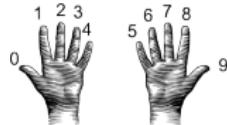
- Set sum = last digit.
- Multiply next digit by  $2 = 2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by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

Sum starts with: 1

$0 \times 2 = 0$ . Add 0 to sum: 1

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

# Binary to Decimal: Converting Between Bases



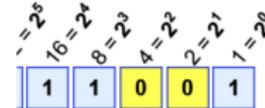
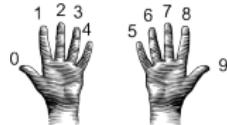
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

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

# Binary to Decimal: Converting Between Bases



$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

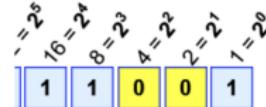
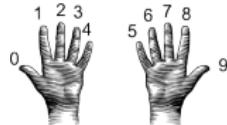
Sum starts with: 1

$0 \times 2 = 0$ . Add 0 to sum: 1

$1 \times 4 = 4$ . Add 4 to sum: 5

$1 \times 8 = 8$ . Add 8 to sum:

# Binary to Decimal: Converting Between Bases



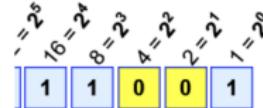
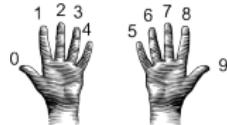
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

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

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

# Binary to Decimal: Converting Between Bases



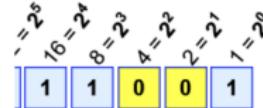
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

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

# Binary to Decimal: Converting Between Bases



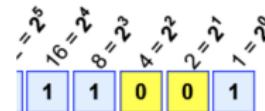
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

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

# Binary to Decimal: Converting Between Bases



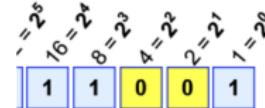
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

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

# Binary to Decimal: Converting Between Bases



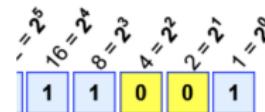
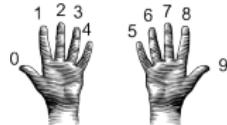
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

Sum starts with: 1  
 $0 \times 2 = 0$ . Add 0 to sum: 1  
 $1 \times 4 = 4$ . Add 4 to sum: 5  
 $1 \times 8 = 8$ . Add 8 to sum: 13  
 $1 \times 16 = 16$ . Add 16 to sum: 29  
 $1 \times 32 = 32$ . Add 32 to sum: 61

# Binary to Decimal: Converting Between Bases



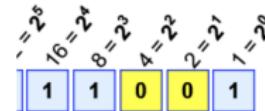
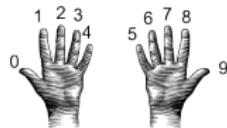
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

Sum starts with:	1
$0 \times 2 = 0$ . Add 0 to sum:	1
$1 \times 4 = 4$ . Add 4 to sum:	5
$1 \times 8 = 8$ . Add 8 to sum:	13
$1 \times 16 = 16$ . Add 16 to sum:	29
$1 \times 32 = 32$ . Add 32 to sum:	61

# Binary to Decimal: Converting Between Bases

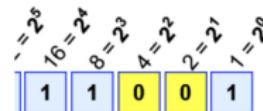
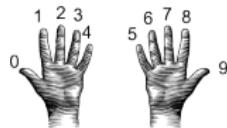


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with:

# Binary to Decimal: Converting Between Bases



Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

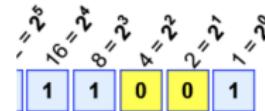
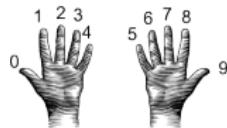
- Example: What is 10100100 in decimal?

Sum starts with:

0

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

# Binary to Decimal: Converting Between Bases



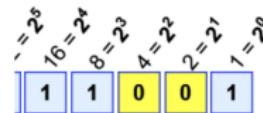
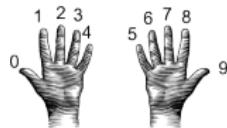
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

# Binary to Decimal: Converting Between Bases



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

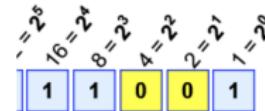
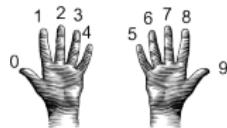
- Example: What is 10100100 in decimal?

Sum starts with: 0

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

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

# Binary to Decimal: Converting Between Bases



Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

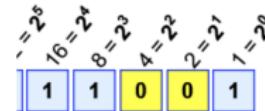
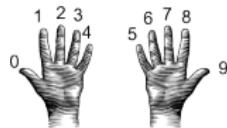
- Example: What is 10100100 in decimal?

Sum starts with: 0

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

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

# Binary to Decimal: Converting Between Bases



Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

- Example: What is 10100100 in decimal?

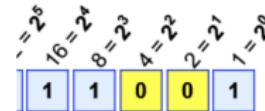
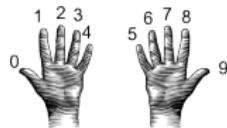
Sum starts with: 0

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

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

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

# Binary to Decimal: Converting Between Bases



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

- Example: What is 10100100 in decimal?

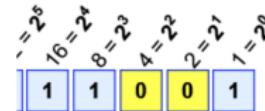
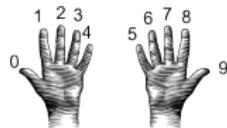
Sum starts with: 0

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

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

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

# Binary to Decimal: Converting Between Bases



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

- Example: What is 10100100 in decimal?

Sum starts with: 0

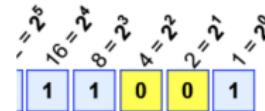
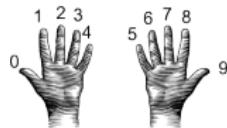
$0 \times 2 = 0$ . Add 0 to sum: 0

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

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

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

# Binary to Decimal: Converting Between Bases



Example:  $1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

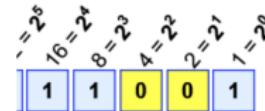
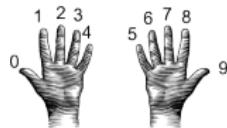
$0 \times 2 = 0$ . Add 0 to sum: 0

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

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

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

# Binary to Decimal: Converting Between Bases



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

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

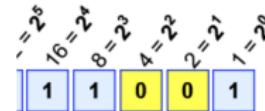
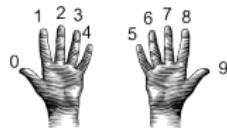
$1 \times 4 = 4$ . Add 4 to sum: 4

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

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

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

# Binary to Decimal: Converting Between Bases



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

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

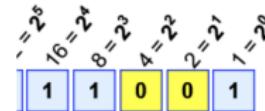
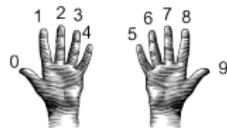
$1 \times 4 = 4$ . Add 4 to sum: 4

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

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

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

# Binary to Decimal: Converting Between Bases



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

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

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

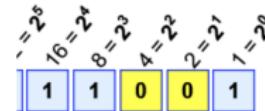
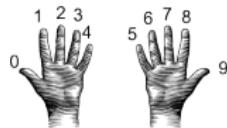
$0 \times 8 = 0$ . Add 0 to sum: 4

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

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

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

# Binary to Decimal: Converting Between Bases



Example:  $1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

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

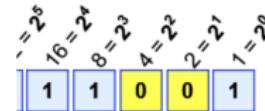
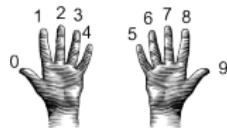
$0 \times 8 = 0$ . Add 0 to sum: 4

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

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

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

# Binary to Decimal: Converting Between Bases

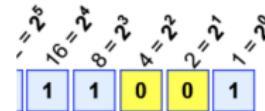
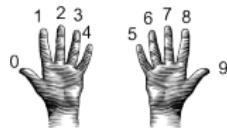


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

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

# Binary to Decimal: Converting Between Bases

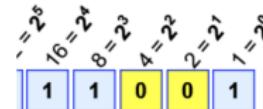
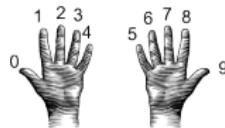


Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 16 + 8 + 4 + 2 + 1 = 25$

- Example: What is 10100100 in decimal?

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

# Binary to Decimal: Converting Between Bases



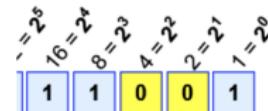
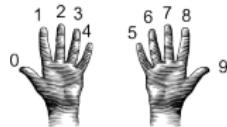
Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

- Example: What is 10100100 in decimal?

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

The answer is 164.

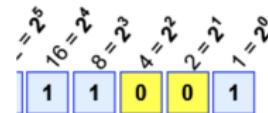
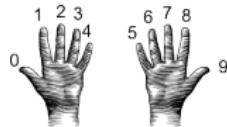
# Design Challenge: Incrementers



$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

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

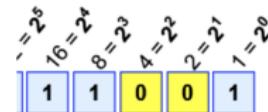
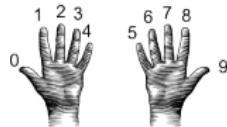
# Design Challenge: Incrementers



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

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

# Design Challenge: Incrementers

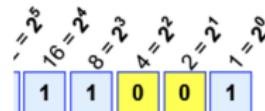
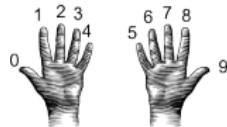


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

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

# Design Challenge: Incrementers



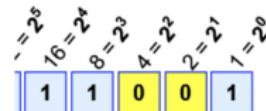
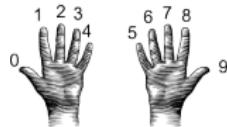
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

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

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

# Design Challenge: Incrementers



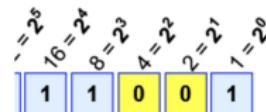
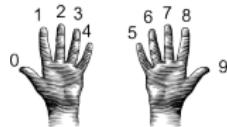
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

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

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

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

# Design Challenge: Incrementers



Example:  $1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 16+8+1 = 25$

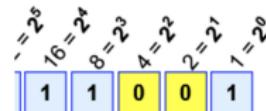
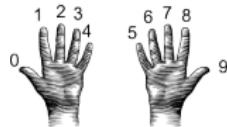
- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

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

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

*Hint: Convert to numbers, increment, and convert back to strings.*

# Design Challenge: Incrementers



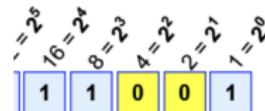
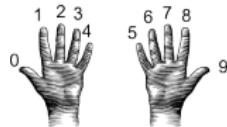
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

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

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

# Design Challenge: Incrementers



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

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

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

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

*Hint: Convert to numbers, increment, and convert back to strings.*

- Challenge: Write an algorithm for incrementing binary numbers.

Example: "1001" → "1010"

# Recap



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

# Recap



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

# 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 during the time of the exam
- The exam format:

# Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on during the time of the exam
- The exam format:
  - ▶ Like a long Lab Quiz, you scroll down to answer all questions.

# Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on during the time of the exam
- The exam format:
  - ▶ Like a long Lab Quiz, you scroll down to answer all questions.
  - ▶ Questions roughly correspond to the 10 parts from old exams, but will appear as a larger number of questions on Gradescope

# Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on during the time of the exam
- The exam format:
  - ▶ Like a long Lab Quiz, you scroll down to answer all questions.
  - ▶ Questions roughly correspond to the 10 parts from old exams, but will appear as a larger number of questions on Gradescope
  - ▶ Questions are variations on the programming assignments, lab exercises, and lecture design challenges.

# Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on during the time of the exam
- The exam format:
  - ▶ Like a long Lab Quiz, you scroll down to answer all questions.
  - ▶ Questions roughly correspond to the 10 parts from old exams, but will appear as a larger number of questions on Gradescope
  - ▶ Questions 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.

# 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.
  - ▶ Avoid scrambling through web searches and waste time during 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.
  - ▶ Avoid scrambling through web searches and waste time during the exam.
- Past exams available on webpage (includes answer keys).

# Class Reminders!



Before next lecture, don't forget to:

- Review this class's Lecture and Lab

# Class Reminders!



Before next lecture, don't forget to:

- Review this class's Lecture and Lab
- Take the Lab Quiz



# Class Reminders!



Before next lecture, don't forget to:

- Review this class's Lecture and Lab
- Take the Lab Quiz 
- Submit this class's programming assignments (programs 50-52)