

# CSci 127: Introduction to Computer Science



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

# Frequently Asked Questions

From email and tutoring.

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

# Frequently Asked Questions

From email and tutoring.

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

*The official final is Monday, 23 May, 9-11am.*

# Frequently Asked Questions

From email and tutoring.

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

*The official final is Monday, 23 May, 9-11am.*

*The early final exam (alternative date) is on Friday, 20 May, 8-10am.*

# Frequently Asked Questions

From email and tutoring.

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

*The official final is Monday, 23 May, 9-11am.*

*The early final exam (alternative date) is on Friday, 20 May, 8-10am.*

*Instead of a review sheet, we have:*

# Frequently Asked Questions

From email and tutoring.

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

*The official final is Monday, 23 May, 9-11am.*

*The early final exam (alternative date) is on Friday, 20 May, 8-10am.*

*Instead of a review sheet, we have:*

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

# Frequently Asked Questions

From email and tutoring.

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

*The official final is Monday, 23 May, 9-11am.*

*The early final exam (alternative date) is on Friday, 20 May, 8-10am.*

*Instead of a review sheet, we have:*

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

# Frequently Asked Questions

From email and tutoring.

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

*The official final is Monday, 23 May, 9-11am.*

*The early final exam (alternative date) is on Friday, 20 May, 8-10am.*

*Instead of a review sheet, we have:*

- ▶ *All previous final exams (and answer keys) on the website.*
- ▶ *UTAs in drop-in tutoring happy to review concepts and old exam questions.*
- ▶ *There will be opportunity for practice during our last meeting on 17 May.*

# 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.
- Stop when found, or the end of list is reached.

# Today's Topics



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

# Python & Circuits Review: 10 Weeks in 10 Minutes



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

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

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

- Introduced comments & print():

```
#Name: Thomas Hunter
```

← These lines are comments

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

← (for us, not computer to read)

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

← (this one also)

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

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

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

- Introduced comments & print():

```
#Name: Thomas Hunter
```

← These lines are comments

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

← (for us, not computer to read)

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

← (this one also)

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

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

- As well as definite loops & the turtle package:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

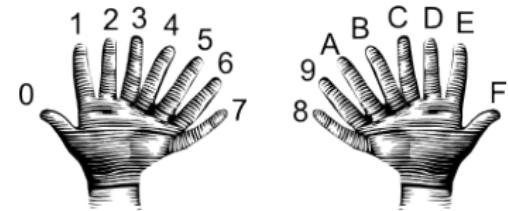
## Week 2: variables, data types, more on loops & range()

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

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

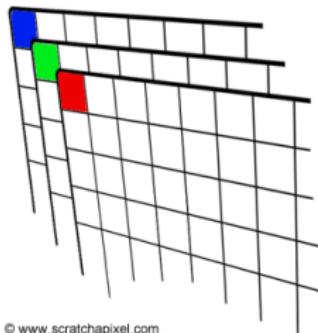
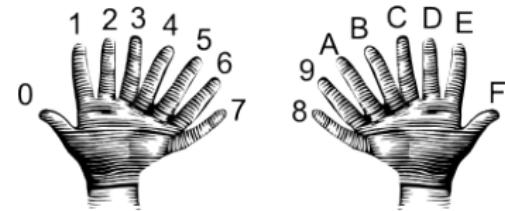
# Week 3: colors, hex, slices, numpy & images

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



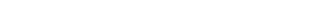
# Week 3: colors, hex, slices, numpy & images

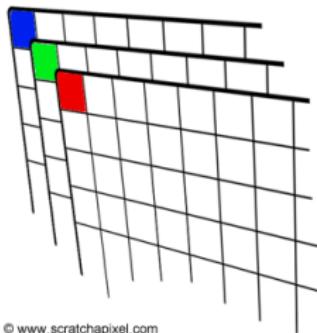
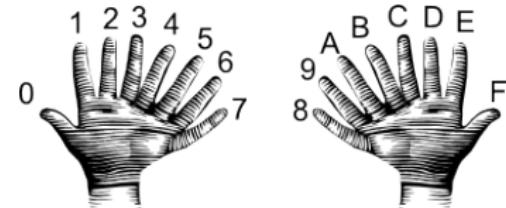
Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



© www.scratchapixel.com

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

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



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

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

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

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

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

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



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



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

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



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

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



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

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

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

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

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

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

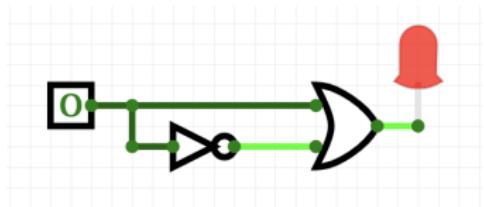
visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
    (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

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

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

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

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



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

Source: [https://en.wikipedia.org/wiki/Demographics\\_of\\_New\\_York\\_City](https://en.wikipedia.org/wiki/Demographics_of_New_York_City).....  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....  
.....  
Year,Borough,Brooklyn,Bronx,Queens,Bronx,Bronx,Staten Island,Totals  
1890,1,231,037,727,718,423  
1871,21843,3623,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,7595  
1810,65544,5534,6083,1753,4573,7534  
1820,123704,11487,8246,2792,4135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,315110,15013,14046,5346,10965,39114  
1850,35545,12850,12850,5346,10965,39114  
1860,613469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59945,5653,51980,39301,1911803  
1890,1363473,65483,5653,51980,39301,1911803  
1900,185093,1166582,152999,200567,67621,2437202  
1910,2233142,1634351,264041,430980,8569,4766803  
1920,2210103,2018354,446707,72021,11651,59148  
1930,1867122,1587128,229354,43583,5831,4930446  
1940,1889924,2498285,1297634,1394711,374441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1698211,2309319,1890591,1465111,191555,7981984  
1970,1539231,2465705,1874473,1472701,195443,7981984  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1203789,378977,7322564  
2000,1537195,2485326,2229379,1332650,419782,8080879  
2010,1583873,2540705,2217722,1385108,447513,8175133  
2015,1444518,2646733,2339159,1459446,474558,8059405

nycHistPop.csv

In Lab 6

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

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

Source: [https://en.wikipedia.org/wiki/Demographics\\_of\\_New\\_York\\_City](https://en.wikipedia.org/wiki/Demographics_of_New_York_City),  
All population figures are consistent with present-day boundaries.....  
Five census after the consolidation of the five boroughs.....  
.....  
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total  
1890,4937,2037,727,788,101  
1891,21843,36231,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,7595  
1810,63541,5544,6240,1755,4543,7595  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,18013,14049,5346,10965,391114  
1850,35549,12891,12891,12891,12891,12891  
1860,813469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59945,5653,51980,33029,1911803  
1890,1367711,66582,61582,5653,51980,33029,1911803  
1900,185093,116582,152999,200567,67621,2437202  
1910,2233142,1634351,28461,430980,8569,4766803  
1920,2210110,2018354,44607,44607,73201,11651,50048  
1930,1867111,1579128,1579128,1579128,1579128,4930446  
1940,1889924,2498285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690101,2319319,1690101,1690101,1690101,781984  
1970,1539231,2465701,1472701,1472701,135443,768460  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1203789,379977,7322564  
2000,1537195,2485326,2229379,1332450,419728,8080879  
2010,1583873,2504705,2277722,1385108,4175133,8175133  
2015,1444018,2436733,2339150,1459446,474558,8059405

nycHistPop.csv

In Lab 6

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

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

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

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City.....  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....  
.....  
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total  
1690,4937,2037,727,788,102  
1771,21843,3623,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,67530,6200,6800,1700,4500,85934  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,19013,14000,5346,10965,391114  
1850,355449,218903,18890,5346,10965,415115  
1860,613469,279122,32003,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59940,5653,51980,33091,1911801  
1890,1367000,70000,65000,58000,33091,2141534  
1900,1850593,116582,152999,200567,67921,2437202  
1910,2233142,1634351,2841,430980,8569,476683  
1920,22461103,2018354,44600,44600,73201,11651,50048  
1930,26671128,2203936,1891325,1891325,15821,4930446  
1940,1889924,2698285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690101,2738175,1550949,1451277,191555,7981984  
1970,1639231,2463701,1472170,1472170,135443,7981984  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1302789,378977,7322564  
2000,1537195,2485326,2223379,1332450,419782,8080879  
2010,1583873,2504705,2272722,1385108,474558,8175133  
2015,1444518,2540733,2339150,1459446,474558,8056405
```

nycHistPop.csv

In Lab 6

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

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

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

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City.....  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....  
.....  
Year,Population  
1698,203,2037,...,727,7181  
1771,21843,36241,...,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,72000,60515,5740,6442,1755,4543,75934  
1820,123704,11487,8246,2792,6135,152056  
1830,20589,20535,9049,3023,7082,242278  
1840,31510,11013,14045,5346,10965,391114  
1850,35549,12890,12890,12890,12890,12890  
1860,813469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59945,5653,51980,33029,1911801  
1890,1385111,71861,6851,6386,59345,25492,174774  
1900,1850993,116582,152999,200567,67921,2437202  
1910,223342,1634351,2841,430980,8569,476683  
1920,2246103,2018354,4460,4460,73203,116582,593446  
1930,2667128,2667128,2667128,2667128,2667128,593446  
1940,1889924,2498285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1696010,1696010,1696010,1696010,1696010,781984  
1970,1539231,1465701,1472701,1472701,135443,798460  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1302789,737997,7723256  
2000,1537195,2485326,2229379,1332450,419782,8080879  
2010,1583873,2504705,2277722,1385108,474558,8175133  
2015,1444018,2646733,2339150,1459446,474558,8175405
```

nycHistPop.csv

In Lab 6

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

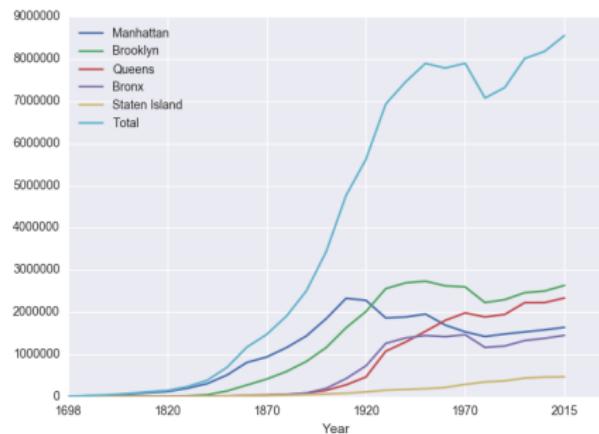
```
import matplotlib.pyplot as plt  
import pandas as pd
```

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

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City.....  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....  
.....  
Year,Borough,Population  
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  
1810,71534,6211,6851,1755,4543,75934  
1820,123704,11187,8246,2792,6135,152056  
1830,202889,20535,9049,3023,7082,242278  
1840,312110,18013,14081,5348,10965,391114  
1850,355441,21800,18851,5851,11515,441134  
1860,813469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33051,1911801  
1890,1384711,72001,6201,57348,33051,2141514  
1900,1850993,1165852,152999,200567,67921,2437202  
1910,233142,1634351,2841,430980,8569,476683  
1920,2210103,2018354,44601,44601,73201,11651,50048  
1930,2667103,2079128,35254,35254,5831,630446  
1940,1889924,2698285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550849,1451277,191555,7891957  
1960,1690101,2738175,1809049,1451277,191555,7891984  
1970,1539231,2465701,1472701,1235443,7891984  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1320789,378977,7322564  
2000,1537195,2485326,2229379,1332450,419728,8080879  
2010,1583873,2504705,2216722,1385108,419728,8175133  
2015,1444518,2636733,2339150,1459446,474558,8059405
```

nycHistPop.csv

In Lab 6



# Week 7: functions

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

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

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

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

# Week 7: functions

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

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

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

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

# Week 7: functions

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

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

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

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

# Week 7: functions

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

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

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

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

# Week 7: functions

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

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

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

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

# Week 7: functions

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

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

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

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

# Week 7: functions

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

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

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

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

# Week 8: function parameters, github

- Functions can have **input parameters**.

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

# Week 8: function parameters, github

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

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

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

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

# Week 8: function parameters, github

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

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

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

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

# Week 8: function parameters, github

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

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

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

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

# Week 8: function parameters, github

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

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

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

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

# Week 8: function parameters, github

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

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

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

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

# Week 9: top-down design, folium, loops, and random()



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

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

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

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

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

```
import turtle
import random

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

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

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

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

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

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

```
import turtle
import random

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

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

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

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

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

```
import turtle
import random

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

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

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

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

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

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

```
import turtle
import random

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

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

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

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

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

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

```
import turtle
import random

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

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

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

# Python & Circuits Review: 10 Weeks in 10 Minutes



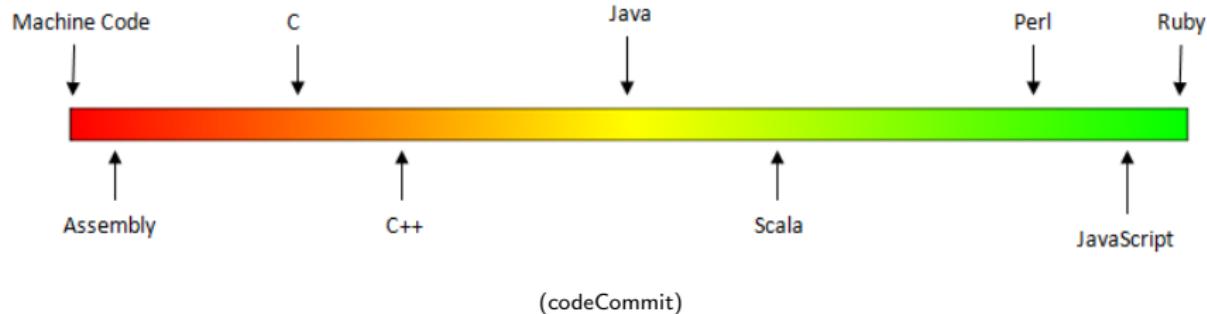
- Input/Output (I/O): `input()` and `print()`; pandas for CSV files
- Types:
  - ▶ Primitive: `int`, `float`, `bool`, `string`;
  - ▶ Container: lists (but not dictionaries/hashes or tuples)
- Objects: turtles (used but did not design our own)
- Loops: definite & indefinite
- Conditionals: `if-elif-else`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

# Today's Topics



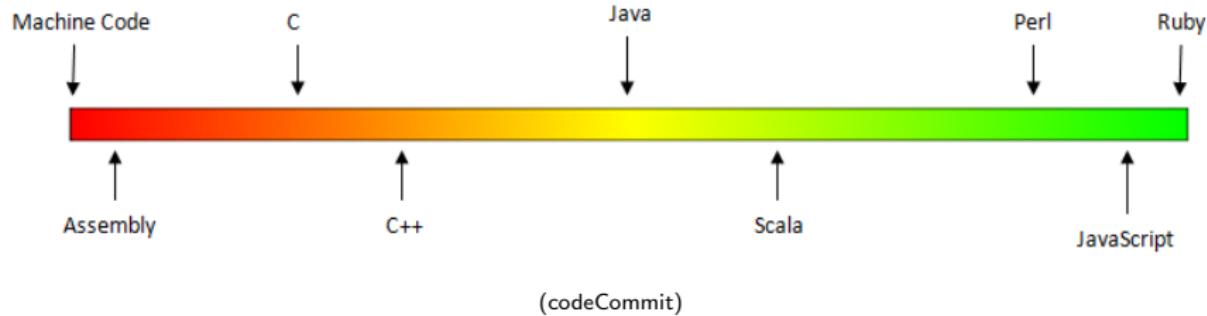
- Design Patterns: Searching
- Python Recap
- **Machine Language**
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic

# Low-Level vs. High-Level Languages



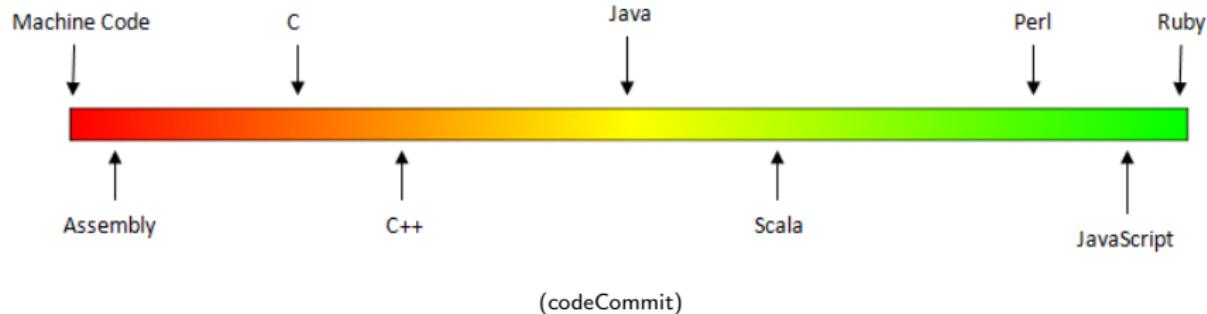
- Can view programming languages on a continuum.

# Low-Level vs. High-Level Languages



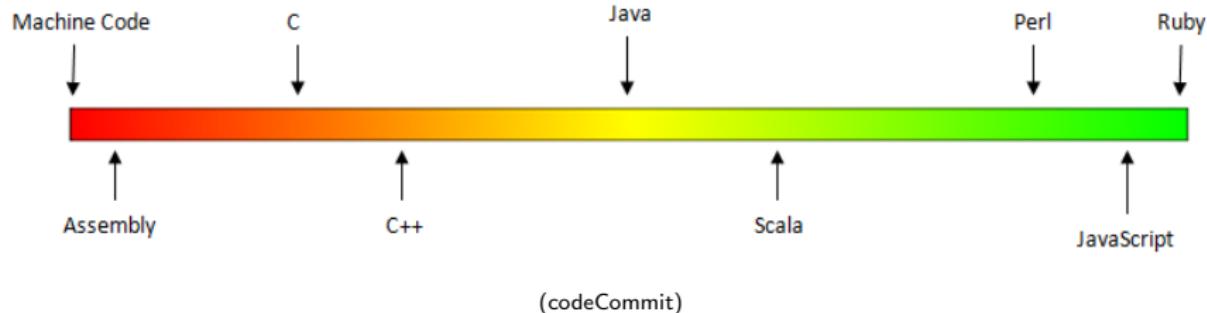
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

# Low-Level vs. High-Level Languages



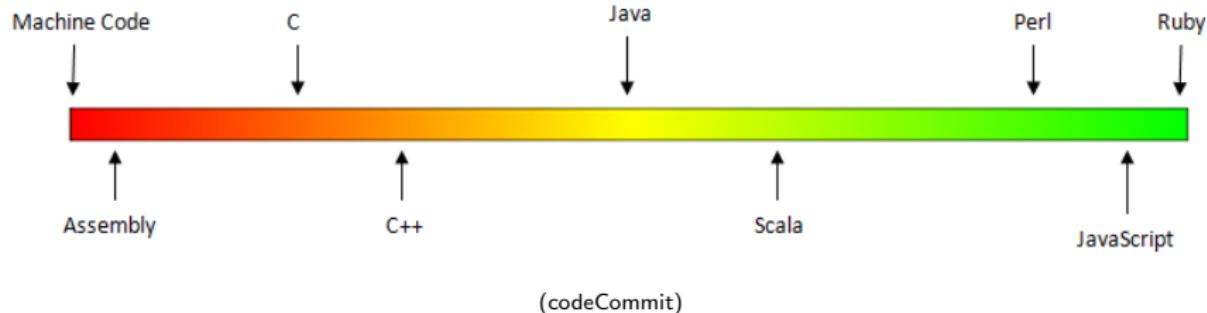
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

# 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 so verdeckt ist, dass er nicht ohne spezielle Methoden gelesen werden kann. Er besteht aus einer Reihe von Zeichen, die nicht im normalen Alphabet oder in einem anderen standardisierten Code angeordnet sind. Der Sinn des Textes kann nur durch den Benutzer erkannt werden, der die Methoden kennt, um den Code zu entschlüsseln.

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

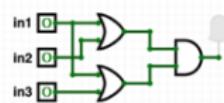


Data  
&  
Instructions

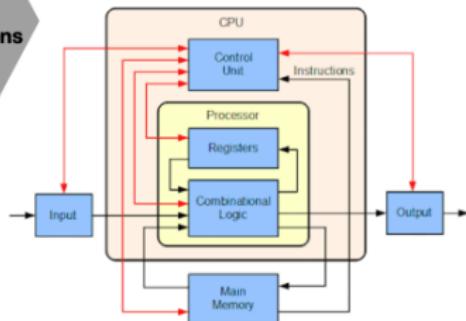


0110100011100100110000101  
1100100011010011011011011  
00111100001101000101011  
001011010000100100011111  
11010101110100011101011  
00100101010110011001000  
01001010101001001000001  
1110010000011010011101  
011000101001100011010101  
010000100000001100010000  
01100101010011001001011  
100101000000110110011101  
01100110010001100100000  
011000100000000010000000  
111000100000000000000000  
01100011010111011001  
000111010101110110010001  
01101101011001101001  
0111001000110111011011  
00100100000100010001001  
0010000001101000101011100  
00010... 10010001100100000  
01101000110111011001  
11010101011001101001  
01110010010111001000101  
000000110110001101100011

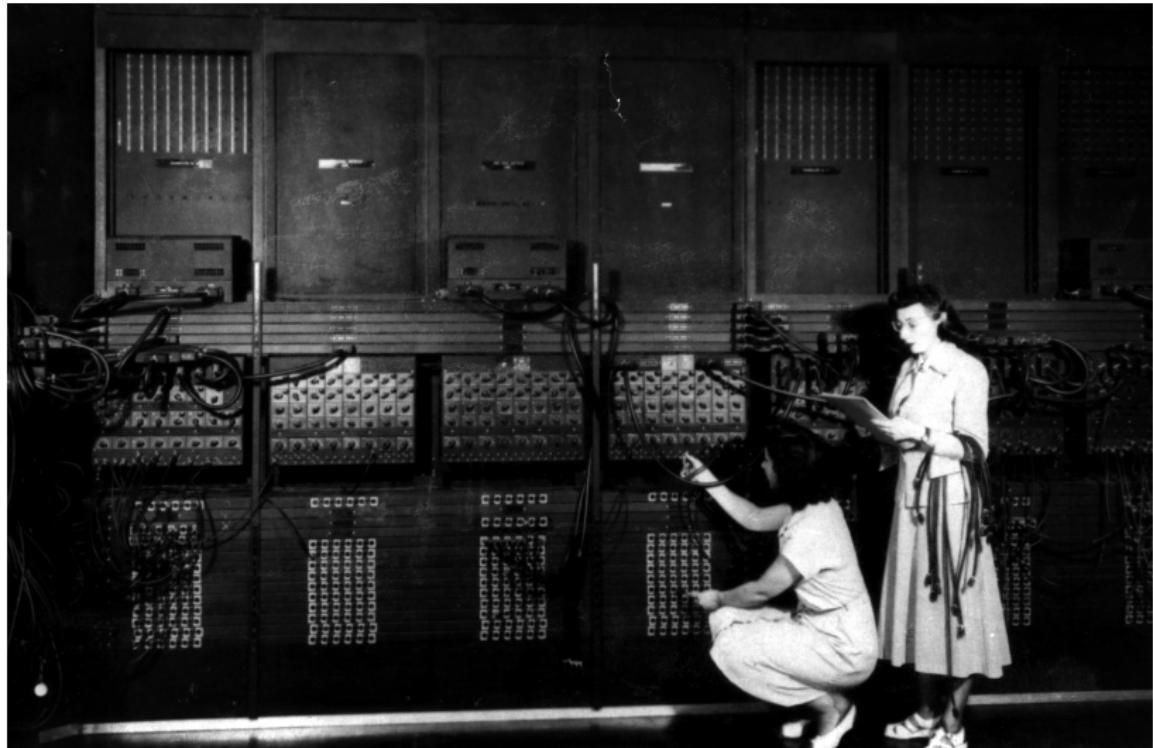
```
def totalWithTax(Food,tip):  
    total = 8  
    tax = 0.0875  
    total = Food + food * tax  
    total = total + tip  
    return(total)
```



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



# Machine Language



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

# Machine Language

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

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

BREAK

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

(wiki)

# Machine Language

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

A screenshot of a terminal window displaying assembly code and its corresponding binary output. The assembly code includes instructions like REP #438, CLD, SEI, LDA #1234, ADC #4324, STA #17F03, CLB, SEC, SBC, and a BREAK instruction. Below the assembly code, the binary representation is shown in two columns: PC and M[PC]. The PC column lists addresses 00000000, 00100000, 0000 0000, 0002, CFFF, 0000, 00, and 2000. The M[PC] column shows the binary values 00 00100000 5555 0000 0002 CFFF 0000 00. A horizontal scroll bar is visible at the bottom of the terminal window.

(wiki)

# Machine Language

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



The screenshot shows a WeMIPS assembly editor interface. The assembly code area contains:

```
    .C2 30    REP $#30
    .B1    CLD
    .SEI
    .LDA #1234
    .LDI 21,43  #4324
    .STW 03,7F,01 #017F03
    .CLD
    .SER #38
    .BRK
    .J 30      #38
    .B012

    PB PC MUw#012C A X Y SP DP R8
    : 00 E012 00110000 0000 0000 0002 CFFF 0000 00
    & 2000

BREAK.
```

The binary code area shows the assembly code mapped to memory addresses:

Address	Value
00E012	00110000 0000 0000 0002 CFFF 0000 00
002000	00110000 5555 0000 0002 CFFF 0000 00
007FF0	55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00

(wiki)

# Machine Language



The screenshot shows a computer interface for writing assembly code. At the top, there's a status bar with 'File' and 'Edit' tabs. Below that is a code editor window containing assembly instructions:

```
    .C2 3B    REP #3B
    .B 1B    CLD
    .B 29E3    SEI
    .B 34 12    LDA #1234
    .B 21 43    ADC #4321
    .B F8 01    STA #01F801
    .B 7F 03    CLD
    .B 30    SEP #30
    .B 00    PSH
    .B 2012    ROR
```

Below the code editor is a register dump window titled 'Registers'. It lists various registers with their current values:

PC	MAR	MDR	A	X	Y	SP	DP	R0
00 E012	00110000	0000 0000 0002 CFFF 0000 00						
29E8								

At the bottom of the register dump, there's a 'BREAK' button.

(wiki)

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

# Machine Language



(wiki)

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

# "Hello World!" in Simplified Machine Language

Line: 3 Go!

Show/Hide Demos

User Guide | Unit Tests | Docs

Addition Doubler Stav Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

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

Step

Run

Enable auto switching

S

T

A

V

Stack

Log

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

(WeMIPS)



# WeMIPS

User | 3 | Dat Show White 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 # $H
2    ADDI $t1, $zero, 101 # $e
3    ADDI $t2, $zero, 101 # $l
4    ADDI $t3, $zero, 108 # $o
5    ADDI $t4, $zero, 108 # $w
6    ADDI $t5, $zero, 108 # $r
7    ADDI $t6, $zero, 108 # $l
8    ADDI $t7, $zero, 108 # $d
9    ADDI $t8, $zero, 101 # $n
10   ADDI $t9, $zero, 101 # $e
11   ADDI $t10, $zero, 101 # $l
12   ADDI $t11, $zero, 41 # $a
13   ADDI $t12, $zero, 32 # $p
14   ADDI $t13, $zero, 32 # $a
15   ADDI $t14, $zero, 32 # $h
16   ADDI $t15, $zero, 32 # $e
17   ADDI $t16, $zero, 32 # $l
18   ADDI $t17, $zero, 32 # $o
19   ADDI $t18, $zero, 32 # $w
20   ADDI $t19, $zero, 32 # $r
21   ADDI $t20, $zero, 32 # $l
22   ADDI $t21, $zero, 32 # $d
23   ADDI $t22, $zero, 32 # $n
24   ADDI $t23, $zero, 32 # $e
25   ADDI $t24, $zero, 32 # $l
26   ADDI $t25, $zero, 33 # $i
27   ADDI $t26, $zero, 32 # $l
28   ADDI $t27, $zero, 32 # $o
29   ADDI $t28, $zero, 32 # $w
30   ADDI $t29, $zero, 32 # $r
31   ADDI $t30, $zero, 4 # 4 is for print string
32   ADDI $t31, $zero, 0 # point to the log
33   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)

# MIPS Commands

The screenshot shows a MIPS assembly debugger interface. At the top, there's a menu bar with 'File', 'Edit', 'Run', 'Help', 'ShowHide Demo', and tabs for 'Addition', 'Calculator', 'Btav', 'Loop', 'Stack Test', 'Hello World', 'Code Gen', 'Save String', 'Interactive', 'Binary', 'Decimal', 'Hexadecimal', and 'Binary'. Below the tabs are buttons for 'Debug' and 'Run'. On the right, there are links for 'User Guide', 'Unit Tests', and 'Docs'. The main area has tabs for 'Step', 'Run', and 'Log'. The 'Step' tab is selected. It displays assembly code:

```
1 # Show "Hello world" at the top of the stack
2 .data
3 .text
4 .globl _start
5 .type _start, @function
6 _start:
7     addiu $sp,$sp,-16      # allocate space for stack frame
8     addiu $t0,$zero,100      # set base pointer to stack
9     addiu $t0,$t0,100        # set stack pointer to base + 100
10    addiu $t1,$zero,100      # set temporary register to 100
11    addiu $t1,$t1,100        # set temporary register to 200
12    addiu $t2,$zero,100      # set temporary register to 100
13    addiu $t2,$t2,100        # set temporary register to 200
14    addiu $t3,$zero,100      # set temporary register to 100
15    addiu $t3,$t3,100        # set temporary register to 200
16    addiu $t4,$zero,100      # set temporary register to 100
17    addiu $t4,$t4,100        # set temporary register to 200
18    addiu $t5,$zero,100      # set temporary register to 100
19    addiu $t5,$t5,100        # set temporary register to 200
20    addiu $t6,$zero,100      # set temporary register to 100
21    addiu $t6,$t6,100        # set temporary register to 200
22    addiu $t7,$zero,100      # set temporary register to 100
23    addiu $t7,$t7,100        # set temporary register to 200
24    addiu $t8,$zero,100      # set temporary register to 100
25    addiu $t8,$t8,100        # set temporary register to 200
26    addiu $t9,$zero,100      # set temporary register to 100
27    addiu $t9,$t9,100        # set temporary register to 200
28    addiu $t10,$zero,100      # set temporary register to 100
29    addiu $t10,$t10,100       # set temporary register to 200
30    addiu $t11,$zero,100      # set temporary register to 100
31    addiu $t11,$t11,100       # set temporary register to 200
32    syscall                # print to the log
```

Below the assembly code, the 'Registers' window shows the following values:

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 ShowMIPS IDE interface. At the top, there are tabs for "ShowMIPS Demo", "User Guide", "Unit Tests", and "Docs". Below the tabs are several menu items: "Addition Counter", "Bitop", "Looper", "Stack Test", "Hello World", "Code Gen Save String", "Interactive", "Binary Decimal", "Decimal Binary", and "Debug". The "Debug" tab is selected. In the center, there is a text area containing assembly code for a "Hello World" program. At the bottom right of the text area is a "Run" button with the text "Enable auto switching". To the right of the text area is a table titled "Registers" with columns for S, T, A, V, Stack, and Log. The table lists the following register values:

S	T	A	V	Stack	Log
\$0	13				
\$1	9				
\$2	22				
\$3	66				
\$4	61				
\$5	807				
\$6	418				
\$7					

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

# MIPS Commands

The screenshot shows the ShowMIPS IDE interface. At the top, there's a menu bar with 'File', 'Edit', 'Run', 'Help', 'User Guide', 'Unit Tests', and 'Doc'. Below the menu is a toolbar with buttons for 'Addition Counter', 'Itiva', 'Looper', 'Stack Test', 'Hello World', 'Code Gen Save String', 'Interactive', 'Binary Decimal', 'Decimal Binary', and 'Debug'. The main area contains assembly code for a 'Hello World' program. The code includes instructions like `ADDI`, `ADD`, `SUB`, `BEQZ`, `BNEZ`, and `JAL` with various immediate values and labels. A log window at the bottom shows the execution of the program. On the right, there's a register dump table with columns for \$, T, A, V, Stack, and Log.

\$	T	A	V	Stack	Log
s0	10				
s1	9				
s2	8				
s3	7				
s4	6				
s5	5				
s6	807				
s7	418				

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

# MIPS Commands

The screenshot shows a window titled "StackFrame Demo". At the top, there are tabs for "User", "File", "Run", "Help", "About", and "Exit". Below the tabs are buttons for "Addition", "Calculator", "ItRev", "Looper", "Stack Test", "Hello World", "Code Gen Save String", "Interactive", "Binary Decimal", "Decimal Binary", and "Debug". The "Debug" tab is selected. On the right side of the window, there is a status bar with "User Guide | Unit Tests | Doc" and a "Stop" button.

The main area 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    li $t0, 0x484f4d4c # 'Hello'
6    addi $t1, $t0, 0x0 # $t0 = &Hello
7    addi $t2, $t0, 0x4 # $t0 = Hello + 4
8    addi $t3, $t0, 0x8 # $t0 = Hello + 8
9    addi $t4, $t0, 0xc # $t0 = Hello + 12
10   addi $t5, $t0, 0x10 # $t0 = Hello + 16
11   addi $t6, $t0, 0x14 # $t0 = Hello + 20
12   addi $t7, $t0, 0x18 # $t0 = Hello + 24
13   addi $t8, $t0, 0x1c # $t0 = Hello + 28
14   addi $t9, $t0, 0x20 # $t0 = Hello + 32
15   addi $t10, $t0, 0x24 # $t0 = Hello + 36
16   addi $t11, $t0, 0x28 # $t0 = Hello + 40
17   addi $t12, $t0, 0x2c # $t0 = Hello + 44
18   addi $t13, $t0, 0x30 # $t0 = Hello + 48
19   addi $t14, $t0, 0x34 # $t0 = Hello + 52
20   addi $t15, $t0, 0x38 # $t0 = Hello + 56
21   addi $t16, $t0, 0x3c # $t0 = Hello + 60
22   addi $t17, $t0, 0x40 # $t0 = Hello + 64
23   addi $t18, $t0, 0x44 # $t0 = Hello + 68
24   addi $t19, $t0, 0x48 # $t0 = Hello + 72
25   addi $t20, $t0, 0x4c # $t0 = Hello + 76
26   addi $t21, $t0, 0x50 # $t0 = Hello + 80
27   addi $t22, $t0, 0x54 # $t0 = Hello + 84
28   addi $t23, $t0, 0x58 # $t0 = Hello + 88
29   addi $t24, $t0, 0x5c # $t0 = Hello + 92
30   addi $t25, $t0, 0x60 # $t0 = Hello + 96
31   addi $t26, $t0, 0x64 # $t0 = Hello + 100
32   syscall # print to the log
```

The right pane shows a table of registers:

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

# MIPS Commands



The screenshot shows the StackFrame Demo application interface. At the top, there are tabs for User, ShowFrame Demo, and other developer tools. Below the tabs are buttons for Addition, Decoder, Itav, Looper, Stack Test, Hello World, Code Gen, Save String, Interactive, Binary, Decimal, and Debug. The main area contains assembly code:

```
# 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 16 bytes on stack
6        addiu   $t0, $zero, 111 # $t0 = 111
7        addiu   $t1, $zero, 128 # $t1 = 128
8        addiu   $t2, $zero, 128 # $t2 = 128
9        addiu   $t3, $zero, 128 # $t3 = 128
10       addiu   $t4, $zero, 128 # $t4 = 128
11       addiu   $t5, $zero, 128 # $t5 = 128
12       addiu   $t6, $zero, 128 # $t6 = 128
13       addiu   $t7, $zero, 128 # $t7 = 128
14       addiu   $t8, $zero, 128 # $t8 = 128
15       addiu   $t9, $zero, 128 # $t9 = 128
16       addiu   $t10, $zero, 128 # $t10 = 128
17       addiu   $t11, $zero, 111 # $t11 = 111
18       addiu   $t12, $zero, 111 # $t12 = 111
19       addiu   $t13, $zero, 111 # $t13 = 111
20       addiu   $t14, $zero, 111 # $t14 = 111
21       addiu   $t15, $zero, 128 # $t15 = 128
22       addiu   $t16, $zero, 128 # $t16 = 128
23       addiu   $t17, $zero, 128 # $t17 = 128
24       addiu   $t18, $zero, 128 # $t18 = 128
25       addiu   $t19, $zero, 128 # $t19 = 128
26       addiu   $t20, $zero, 128 # $t20 = 128
27       addiu   $t21, $zero, 0 # (null)
28       addiu   $t22, $zero, 0 # (null)
29       addiu   $t23, $zero, 0 # (null)
30       addiu   $t24, $zero, 0 # (null)
31       addiu   $t25, $zero, 0 # (null)
32       addiu   $t26, $zero, 0 # print to the log
33       syscall
```

Below the assembly code, there is a table showing register values:

S	T	A	V	Stack	Log
\$0				13	
\$1				9	
\$2				22	
\$3				65	
\$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.

# 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 command buttons: 'Addition Counter', 'Btav', 'Looper', 'Stack Test', 'Hello World', 'Code Gen Save String', 'Interactive', 'Binary Decimal', 'Decimal Binary', and 'Debug'. The main area contains assembly code and a register dump.

**Assembly Code:**

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

**Registers:**

S	T	A	V	Stack	Log
s0				10	
s1				9	
s2				8	
s3				7	
s4				6	
s5				5	
s6				4	
s7				3	
s8				2	
s9				1	
s10				0	
s11				418	
s12					
s13					
s14					
s15					
s16					
s17					
s18					
s19					
s20					
s21					
s22					
s23					
s24					
s25					

- **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

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

# MIPS Commands

The screenshot shows the StackFrame Demo application window. At the top, there are tabs for User, Show, and Go, along with links for ShowFrame Demo, Addition, Divider, ItInv, Looper, Stack Test, Hello World, Code Gen, Save String, Interactive, Binary, Decimal, Debug, and Help. Below the tabs is a toolbar with buttons for Addition, Divider, ItInv, Looper, Stack Test, Hello World, Code Gen, Save String, Interactive, Binary, Decimal, and Debug.

The main area displays assembly code:

```
1 # Shows "Hello world" at the top of the stack
2 .data
3 msg: .asciiz "Hello world\n"
4 .text
5 addi $t0, $zero, 101 # $t0 = 101
6 addi $t1, $zero, 100 # $t1 = 100
7 addi $t2, $zero, 100 # $t2 = 100
8 addi $t3, $zero, 100 # $t3 = 100
9 addi $t4, $zero, 100 # $t4 = 100
10 addi $t5, $zero, 100 # $t5 = 100
11 addi $t6, $zero, 100 # $t6 = 100
12 addi $t7, $zero, 100 # $t7 = 100
13 addi $t8, $zero, 100 # $t8 = 100
14 addi $t9, $zero, 100 # $t9 = 100
15 addi $t10, $zero, 100 # $t10 = 100
16 addi $t11, $zero, 100 # $t11 = 100
17 addi $t12, $zero, 100 # $t12 = 100
18 addi $t13, $zero, 100 # $t13 = 100
19 addi $t14, $zero, 100 # $t14 = 100
20 addi $t15, $zero, 100 # $t15 = 100
21 addi $t16, $zero, 100 # $t16 = 100
22 addi $t17, $zero, 100 # $t17 = 100
23 addi $t18, $zero, 100 # $t18 = 100
24 addi $t19, $zero, 100 # $t19 = 100
25 addi $t20, $zero, 100 # $t20 = 100
26 addi $t21, $zero, 100 # $t21 = 100
27 addi $t22, $zero, 0 # (call)
28 addi $t23, $zero, 0 # (return)
29 addi $t24, $zero, 4 # $t24 = 4 for print string
30 addi $t25, $zero, 0 # $t25 = 0
31 addi $t26, $zero, 0 # $t26 = 0
32 syscall # print to the log
```

To the right of the assembly code is a register dump table:

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				418	
\$12					
\$13					
\$14					
\$15					
\$16					
\$17					
\$18					
\$19					
\$20					
\$21					
\$22					
\$23					
\$24					
\$25					
\$26					
\$27					
\$28					
\$29					
\$30					
\$31					

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

# MIPS Commands

The screenshot shows the MIPS Simulator interface. At the top, there's a menu bar with tabs like User, File, ShowCode Demo, Addition, Subtraction, Bitwise, Looper, Stack Test, Hello World, Code Gen, Save String, Interactive, Binary, Decimal, Debug, and Help. Below the menu is a toolbar with icons for Addition, Subtraction, Bitwise, Looper, Stack Test, Hello World, Code Gen, Save String, Interactive, Binary, Decimal, and Debug. On the right, there are links for User Guide and Unit Tests.

The main area contains assembly code and register values. The assembly code is as follows:

```
# Shows "Hello world" at the top of the stack
1  LUI $t0, 0x48454C4C      # 'Hello' @ $t0
2  ADDI $t1, $t0, -1           # $t1 = $t0 - 1
3  ADDI $t2, $t0, 1             # $t2 = $t0 + 1
4  ADDI $t3, $t0, 0             # $t3 = $t0 + 0
5  ADDI $t4, $t0, 2             # $t4 = $t0 + 2
6  ADDI $t5, $t0, 3             # $t5 = $t0 + 3
7  ADDI $t6, $t0, 4             # $t6 = $t0 + 4
8  ADDI $t7, $t0, 5             # $t7 = $t0 + 5
9  ADDI $t8, $t0, 6             # $t8 = $t0 + 6
10  ADDI $t9, $t0, 7             # $t9 = $t0 + 7
11  ADDI $t10, $t0, 8            # $t10 = $t0 + 8
12  ADDI $t11, $t0, 9            # $t11 = $t0 + 9
13  ADDI $t12, $t0, 10            # $t12 = $t0 + 10
14  ADDI $t13, $t0, 11            # $t13 = $t0 + 11
15  ADDI $t14, $t0, 12            # $t14 = $t0 + 12
16  ADDI $t15, $t0, 13            # $t15 = $t0 + 13
17  ADDI $t16, $t0, 14            # $t16 = $t0 + 14
18  ADDI $t17, $t0, 15            # $t17 = $t0 + 15
19  ADDI $t18, $t0, 16            # $t18 = $t0 + 16
20  ADDI $t19, $t0, 17            # $t19 = $t0 + 17
21  ADDI $t20, $t0, 18            # $t20 = $t0 + 18
22  ADDI $t21, $t0, 19            # $t21 = $t0 + 19
23  ADDI $t22, $t0, 20            # $t22 = $t0 + 20
24  ADDI $t23, $t0, 21            # $t23 = $t0 + 21
25  ADDI $t24, $t0, 22            # $t24 = $t0 + 22
26  ADDI $t25, $t0, 23            # $t25 = $t0 + 23
27  ADDI $t26, $t0, 24            # $t26 = $t0 + 24
28  ADDI $t27, $t0, 25            # $t27 = $t0 + 25
29  ADDI $t28, $t0, 26            # $t28 = $t0 + 26
30  ADDI $t29, $t0, 27            # $t29 = $t0 + 27
31  ADDI $t30, $t0, 28            # $t30 = $t0 + 28
32  ADDI $t31, $t0, 29            # $t31 = $t0 + 29
# print to the log
33  syscall
```

Below the assembly code is a table titled 'Registers' with columns S, T, A, V, Stack, and Log. The table shows the following register values:

S	T	A	V	Stack	Log
\$0	10				
\$1	9				
\$2	8				
\$3	7				
\$4	6				
\$5	5				
\$6	427				
\$7	418				

The memory dump section shows the stack starting at address \$t0 (0x48454C4C) containing the string "Hello world".

- **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 | Show/Hide Device | User Guide | Unit Tests | Docs

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    .text
2    .globl _start
3    .data
4    _strc: .asciz "Hello world!\n"
5
6    _start:
7    la $t0, _strc
8    li $t1, 72 # N
9    la $t2, _strc
10   lq $t3, 0($t2)
11   addi $t2, $t2, 4($t3)
12   sll $t3, $t3, 32 # (opcode)
13   addi $t2, $t2, 1
14   sll $t3, $t3, 512
15   addi $t2, $t2, 119 # '
16   sll $t3, $t3, 119
17   addi $t2, $t2, 111 # o
18   sll $t3, $t3, 111
19   addi $t2, $t2, 108 # l
20   sll $t3, $t3, 108
21   addi $t2, $t2, 104 # r
22   sll $t3, $t3, 104
23   addi $t2, $t2, 101 # i
24   sll $t3, $t3, 101
25   addi $t2, $t2, 105 # n
26   sll $t3, $t3, 105
27   addi $t2, $t2, 0 # null
28   sll $t3, $t3, 0
29
30   la $t0, _strc
31   addi $t0, $t0, 4 # 4 is for print string
32   addi $t0, $t0, 0 # point to the log
33   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:	695				
s5:	976				
s6:	977				
s7:	418				

(Demo with WeMIPS)

# Today's Topics



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

# Loops & Jumps in Machine Language

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



# Loops & Jumps in Machine Language

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



# Loops & Jumps in Machine Language

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



# Loops & Jumps in Machine Language

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



# Loops & Jumps in Machine Language

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



# Loops & Jumps in Machine Language

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



# Loops & Jumps in Machine Language

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



# Jump Demo

Line: 18 Go!

Show/Hide Demos

User Guide | Unit Tests | Docs

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

(Demo  
with  
WeMIPS)

Step Run  Enable auto switching

S T A V Stack Log

Clear Log

Emulation complete, returning to line 1

abcdefghijklmnopqrstuvwxyz

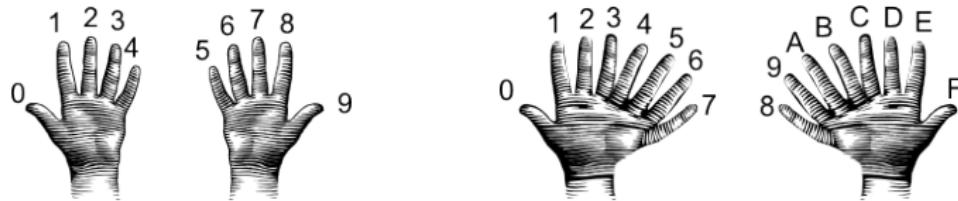


# Today's Topics



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

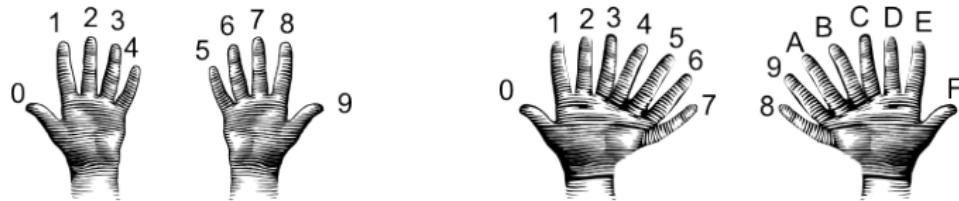
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

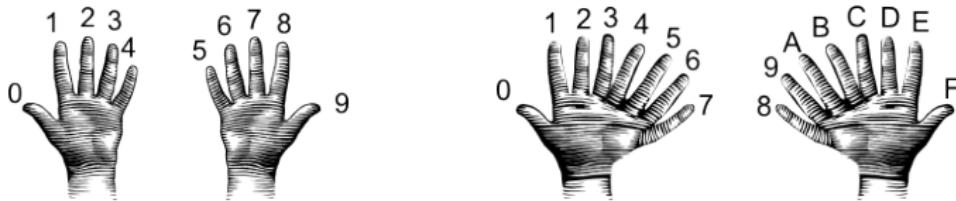
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

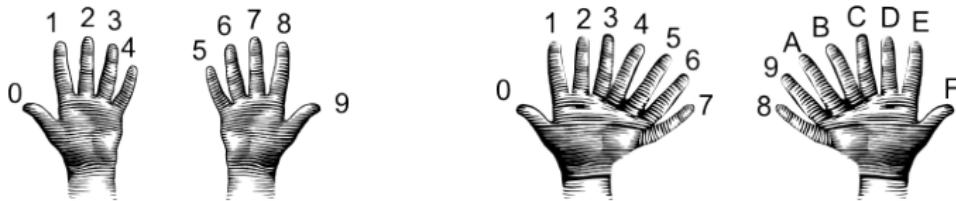
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

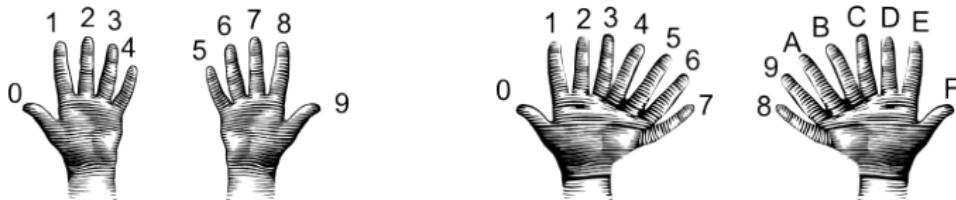
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

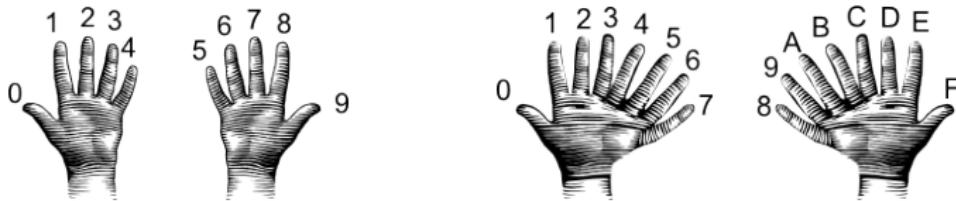
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

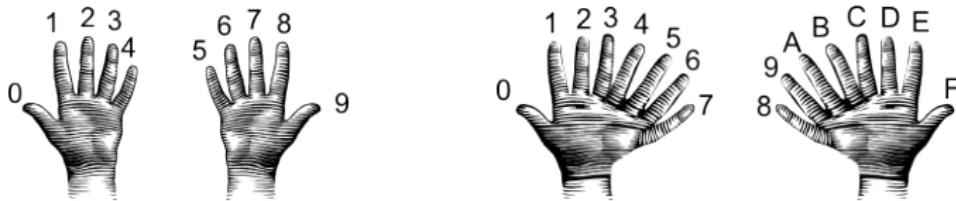
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

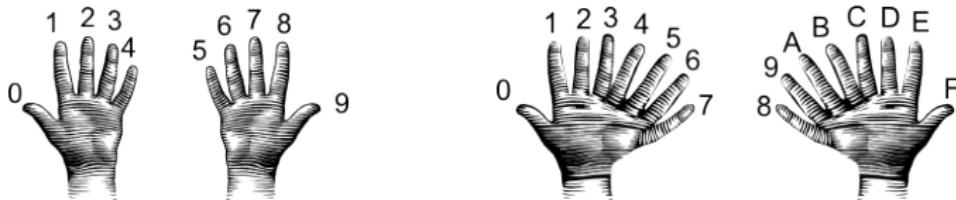
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

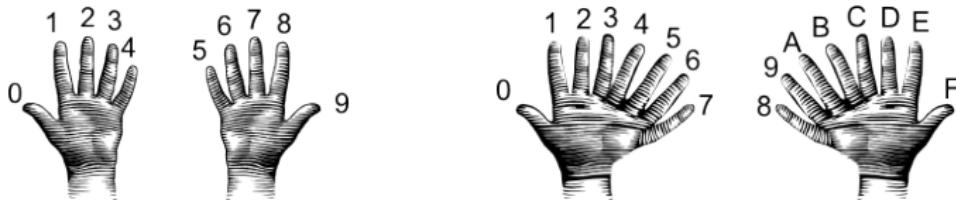
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

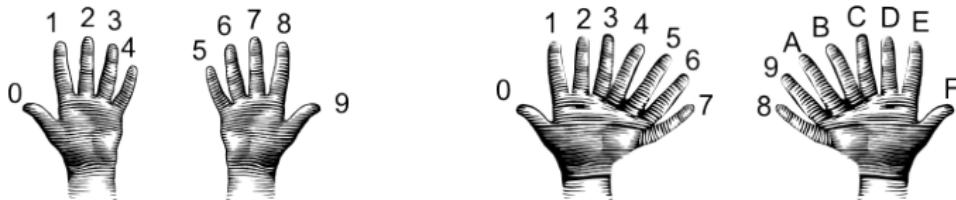
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

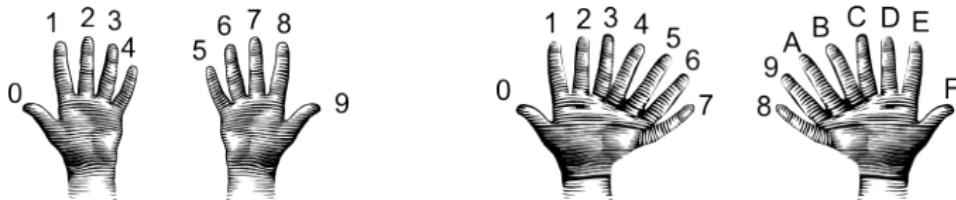
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

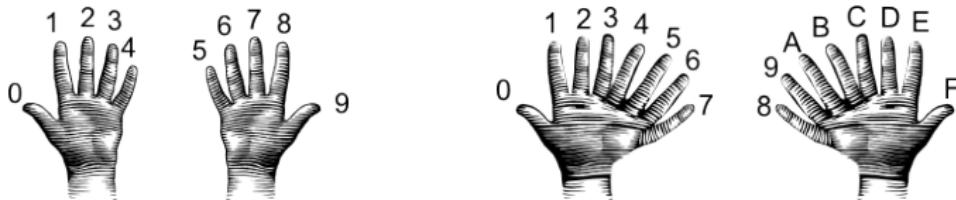
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

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

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

A in decimal digits is 10.

$32 + 10$  is 42.

Answer is 42.

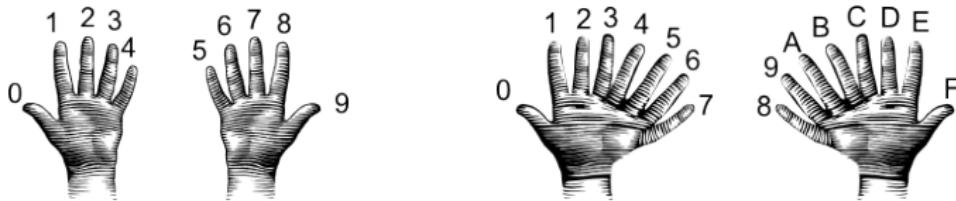
- Example: what is 99 as a decimal number?

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

9 in decimal digits is 9

$144 + 9$  is 153.

# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

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

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

A in decimal digits is 10.

$32 + 10$  is 42.

Answer is 42.

- Example: what is 99 as a decimal number?

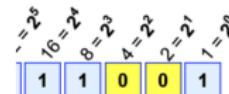
9 in decimal is 9.  $9 \times 16$  is 144.

9 in decimal digits is 9

$144 + 9$  is 153.

Answer is 153.

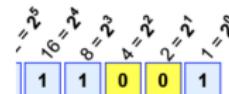
# Decimal to Binary: Converting Between Bases



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

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

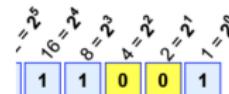
# Decimal to Binary: Converting Between Bases



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

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

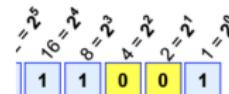
# Decimal to Binary: Converting Between Bases



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

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

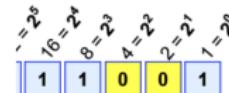
# Decimal to Binary: Converting Between Bases



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

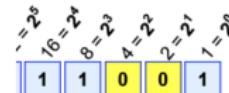
- From decimal to binary:
  - Divide by 128 ( $= 2^7$ ). Quotient is the first digit.
  - Divide remainder by 64 ( $= 2^6$ ). Quotient is the next digit.
  - Divide remainder by 32 ( $= 2^5$ ). Quotient is the next digit.
  - Divide remainder by 16 ( $= 2^4$ ). Quotient is the next digit.

# 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

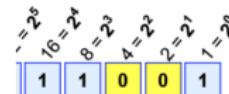


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From decimal to binary:

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

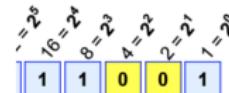
# Decimal to Binary: Converting Between Bases



- From decimal to binary:

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

# Decimal to Binary: Converting Between Bases

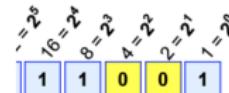


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From decimal to binary:

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

# Decimal to Binary: Converting Between Bases

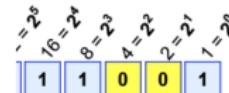


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From decimal to binary:

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

# Decimal to Binary: Converting Between Bases



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

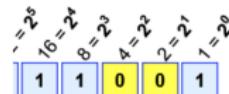
- From decimal to binary:

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

- Example: what is 130 in binary notation?

130/128 is 1 rem 2.

# Decimal to Binary: Converting Between Bases



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

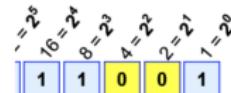
- From decimal to binary:

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

- Example: what is 130 in binary notation?

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

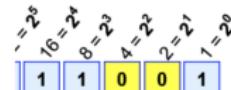
- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient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

- Example: what is 130 in binary notation?

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

2/64 is 0 rem 2.

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

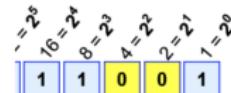
- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient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

- Example: what is 130 in binary notation?

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

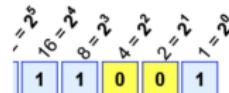
- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient 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. Next digit is 0: 10...

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

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

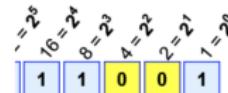
- Example: what is 130 in binary notation?

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

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

2/32 is 0 rem 2.

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

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

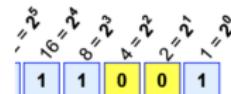
- Example: what is 130 in binary notation?

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

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

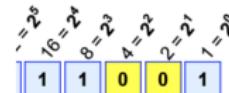
- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient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

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

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

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

# Decimal to Binary: Converting Between Bases



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

- 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.
- Example: what is 130 in binary notation?

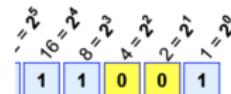
130/128 is 1 rem 2. First digit is 1: 1...

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?

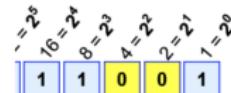
130/128 is 1 rem 2. First digit is 1: 1...

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

- 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?

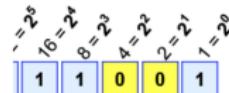
130/128 is 1 rem 2. First digit is 1: 1...

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...

# 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...

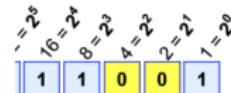
2/64 is 0 rem 2. Next digit is 0: 10...

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.

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

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

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

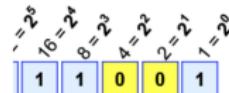
2/64 is 0 rem 2. Next digit is 0: 10...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

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

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

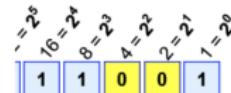
2/64 is 0 rem 2. Next digit is 0: 10...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- 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...

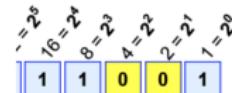
2/32 is 0 rem 2. Next digit is 0: 100...

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

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

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...

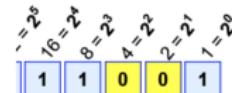
2/32 is 0 rem 2. Next digit is 0: 100...

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

- 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...

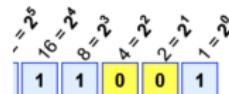
2/32 is 0 rem 2. Next digit is 0: 100...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

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

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

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

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

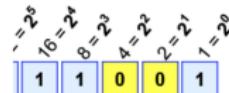
2/16 is 0 rem 2. Next digit is 0: 1000...

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

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

2/2 is 1 rem 0.

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

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

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

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

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

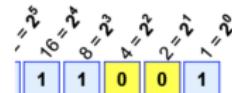
2/16 is 0 rem 2. Next digit is 0: 1000...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- 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...

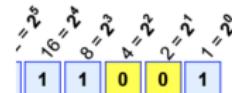
2/16 is 0 rem 2. Next digit is 0: 1000...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

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

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

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

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

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

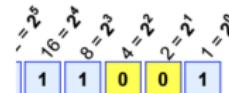
2/8 is 0 rem 2. Next digit is 0: 10000...

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

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

Adding the last remainder: 10000010

# Decimal to Binary: Converting Between Bases



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

- From decimal to binary:

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

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

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

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

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

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

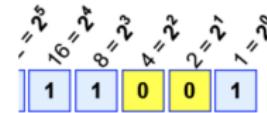
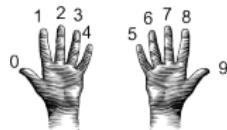
2/4 is 0 remainder 2. Next digit is 0: 100000...

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

Adding the last remainder: 10000010



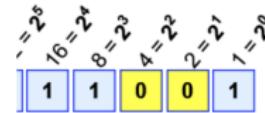
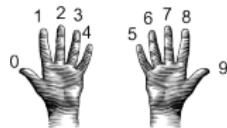
# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

# Decimal to Binary: Converting Between Bases

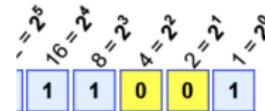
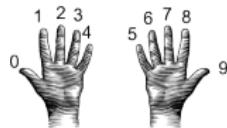


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

$99/128$  is 0 rem 99.

# Decimal to Binary: Converting Between Bases

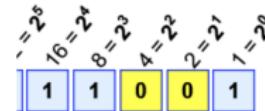
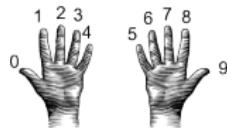


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

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

# Decimal to Binary: Converting Between Bases



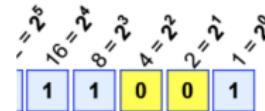
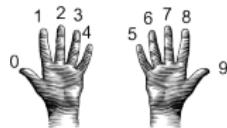
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

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

99/64 is 1 rem 35.

# Decimal to Binary: Converting Between Bases



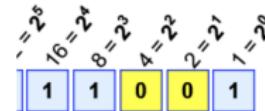
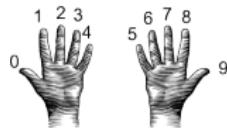
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

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

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

# Decimal to Binary: Converting Between Bases



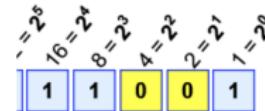
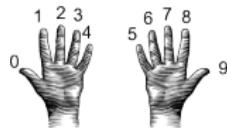
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

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

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

# Decimal to Binary: Converting Between Bases



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

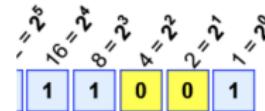
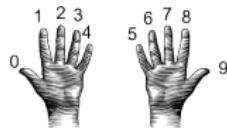
- Example: what is 99 in binary notation?

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

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

35/32 is 1 rem 3.

# Decimal to Binary: Converting Between Bases



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

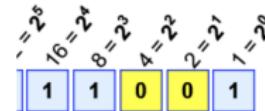
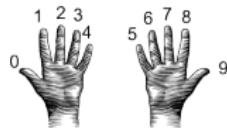
- Example: what is 99 in binary notation?

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

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

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

# Decimal to Binary: Converting Between Bases



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

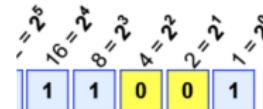
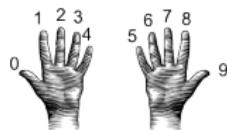
- Example: what is 99 in binary notation?

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?

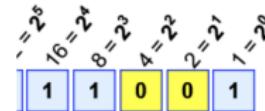
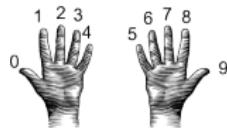
99/128 is 0 rem 99. First digit is 0: 0...

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?

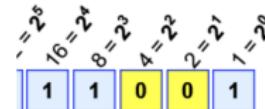
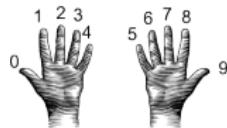
99/128 is 0 rem 99. First digit is 0: 0...

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?

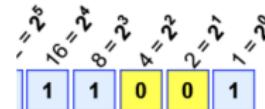
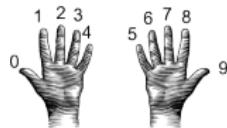
99/128 is 0 rem 99. First digit is 0: 0...

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...

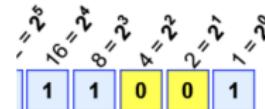
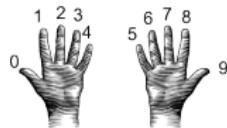
99/64 is 1 rem 35. Next digit is 1: 01...

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...

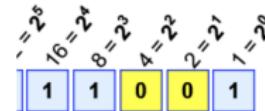
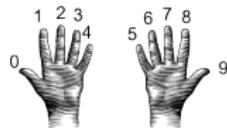
99/64 is 1 rem 35. Next digit is 1: 01...

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

- Example: what is 99 in binary notation?

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

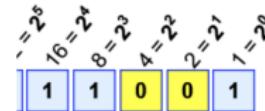
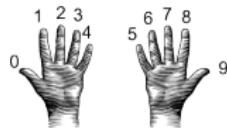
99/64 is 1 rem 35. Next digit is 1: 01...

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...

# Decimal to Binary: Converting Between Bases



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

- 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...

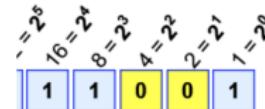
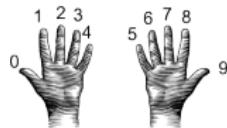
35/32 is 1 rem 3. Next digit is 1: 011...

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

- 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...

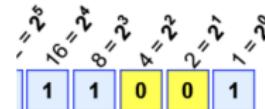
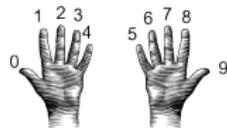
35/32 is 1 rem 3. Next digit is 1: 011...

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:

# 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...

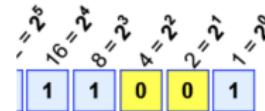
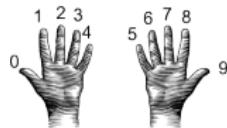
35/32 is 1 rem 3. Next digit is 1: 011...

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...

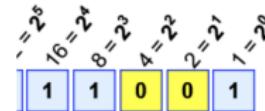
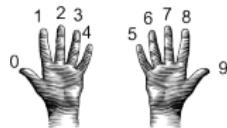
3/16 is 0 rem 3. Next digit is 0: 0110...

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

- 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...

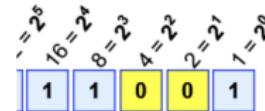
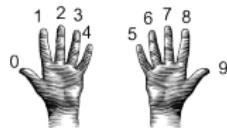
3/16 is 0 rem 3. Next digit is 0: 0110...

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:

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

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

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

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

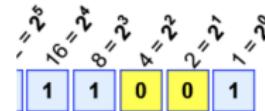
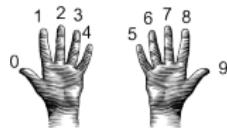
3/16 is 0 rem 3. Next digit is 0: 0110...

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

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

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

# Decimal to Binary: Converting Between Bases



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

- Example: what is 99 in binary notation?

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

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

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

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

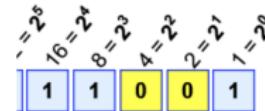
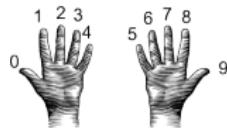
3/8 is 0 rem 3. Next digit is 0: 01100...

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

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

Adding the last remainder: 01100011

# Decimal to Binary: Converting Between Bases



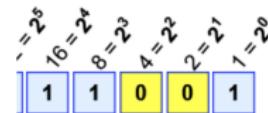
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

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

Answer is 1100011.

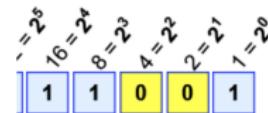
# Binary to Decimal: Converting Between Bases



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

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

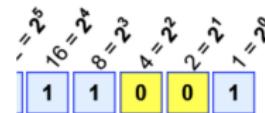
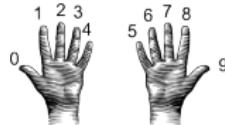
# Binary to Decimal: Converting Between Bases



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

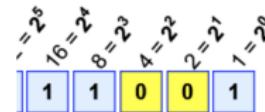
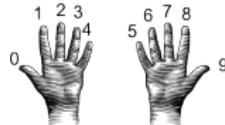


$$\text{Example: } 1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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

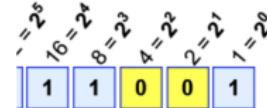
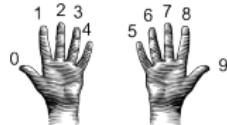


$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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.

# Binary to Decimal: Converting Between Bases

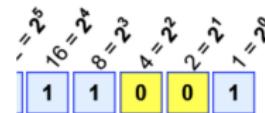


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

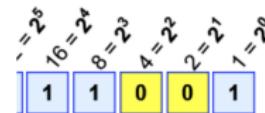


$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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

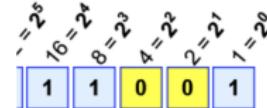


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

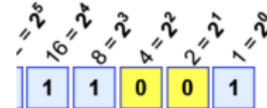


$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

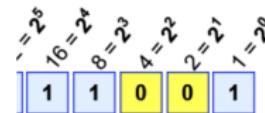
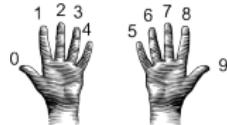


$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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



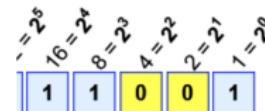
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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



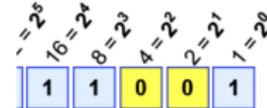
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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



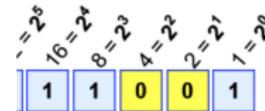
$$\text{Example: } 1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 1 = 25$$

- From binary to decimal:

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

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

# Binary to Decimal: Converting Between Bases



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

- From binary to decimal:

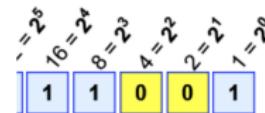
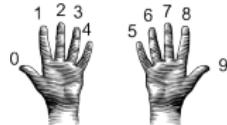
- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $2^2$ . Add to sum.
- Multiply next digit by  $2^3$ . Add to sum.
- Multiply next digit by  $2^4$ . Add to sum.
- Multiply next digit by  $2^5$ . Add to sum.
- Multiply next digit by  $2^6$ . Add to sum.
-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



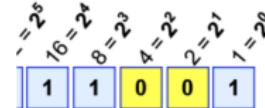
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$$

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

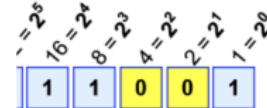
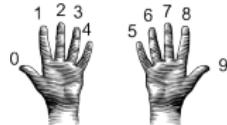
Sum starts with: 1

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

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

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

# Binary to Decimal: Converting Between Bases



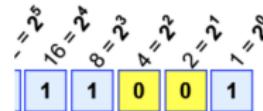
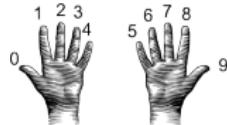
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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: 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



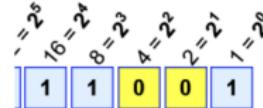
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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  
 $0 \times 2 = 0$ . Add 0 to sum: 1  
 $1 \times 4 = 4$ . Add 4 to sum: 5  
 $1 \times 8 = 8$ . Add 8 to sum: 13  
 $1 \times 16 = 16$ . Add 16 to sum:

# Binary to Decimal: Converting Between Bases



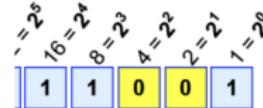
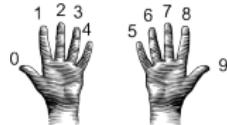
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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

# Binary to Decimal: Converting Between Bases



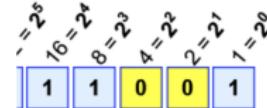
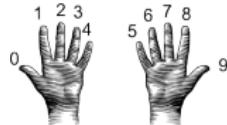
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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  
1\*32 = 32. Add 32 to sum:

# Binary to Decimal: Converting Between Bases



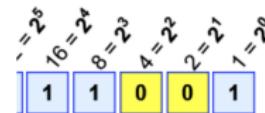
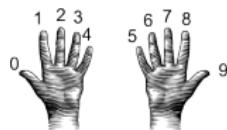
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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



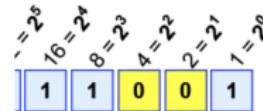
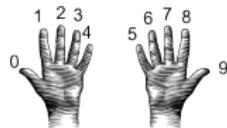
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$$

- 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

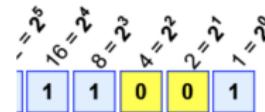
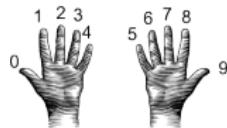


Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with:

# Binary to Decimal: Converting Between Bases



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

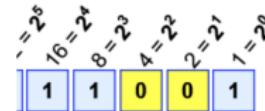
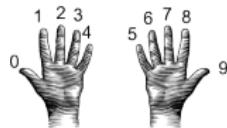
- Example: What is 10100100 in decimal?

Sum starts with:

0

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

# Binary to Decimal: Converting Between Bases



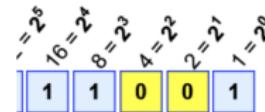
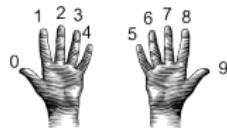
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

# Binary to Decimal: Converting Between Bases



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

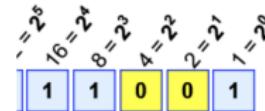
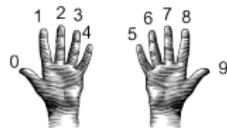
- Example: What is 10100100 in decimal?

Sum starts with: 0

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

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

# Binary to Decimal: Converting Between Bases



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

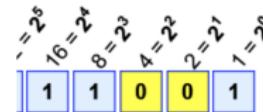
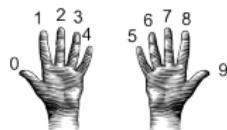
- Example: What is 10100100 in decimal?

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

- Example: What is 10100100 in decimal?

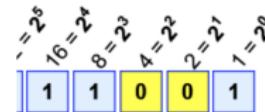
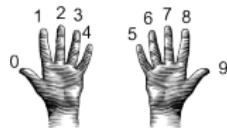
Sum starts with: 0

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

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

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

# Binary to Decimal: Converting Between Bases

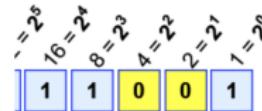
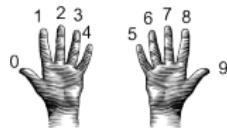


Example:  $1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

# 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

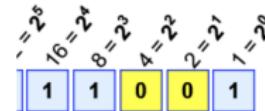
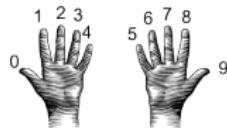
$0 \times 2 = 0$ . Add 0 to sum: 0

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

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

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

# Binary to Decimal: Converting Between Bases



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

- Example: What is 10100100 in decimal?

Sum starts with: 0

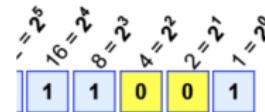
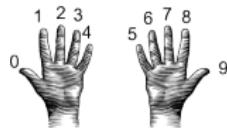
$0 \times 2 = 0$ . Add 0 to sum: 0

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

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

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

# Binary to Decimal: Converting Between Bases



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

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

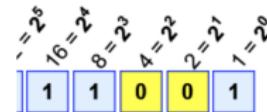
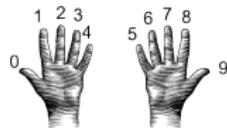
$1 \times 4 = 4$ . Add 4 to sum: 4

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

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

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

# Binary to Decimal: Converting Between Bases



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

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

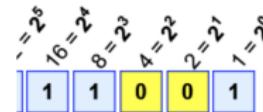
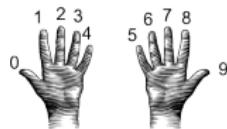
$1 \times 4 = 4$ . Add 4 to sum: 4

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

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

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

# Binary to Decimal: Converting Between Bases



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

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

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

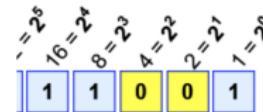
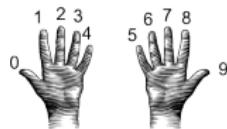
$0 \times 8 = 0$ . Add 0 to sum: 4

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

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

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

# Binary to Decimal: Converting Between Bases



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

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

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

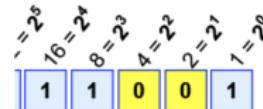
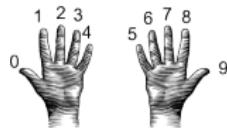
$0 \times 8 = 0$ . Add 0 to sum: 4

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

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

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

# Binary to Decimal: Converting Between Bases

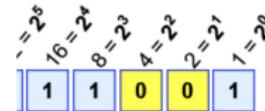
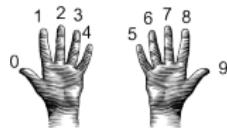


Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

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

# Binary to Decimal: Converting Between Bases

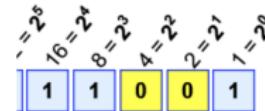
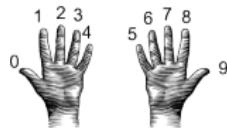


Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

- 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
$0 \times 8 = 0$ . 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



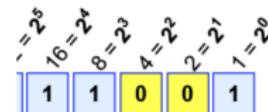
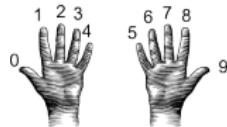
Example:  $1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 16 + 8 + 4 + 0 + 1 = 25$

- 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.

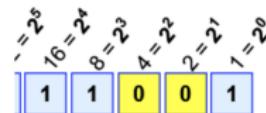
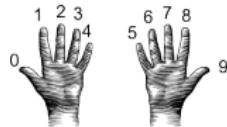
# Design Challenge: Incrementers



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

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

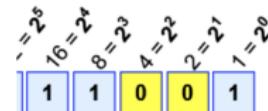
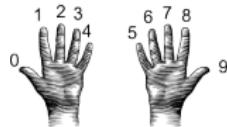
# Design Challenge: Incrementers



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

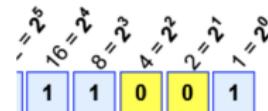
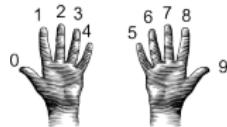


Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

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

# Design Challenge: Incrementers



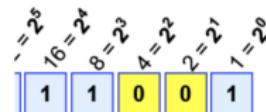
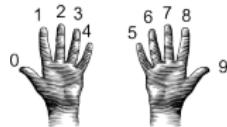
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



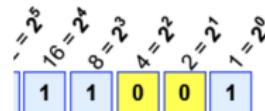
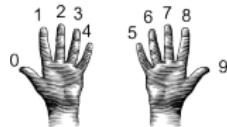
$$\text{Example: } 1 \times 16 + 1 \times 8 + 1 \times 1 = 16+8+1 = 25$$

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

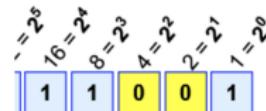
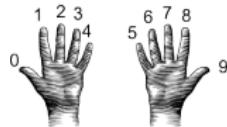
- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

```
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



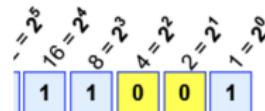
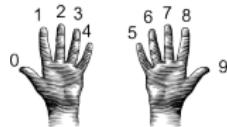
Example:  $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

# Recap



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

# Recap



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

# Final Overview: Format

- The exam is 2 hours long.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.

# Final Overview: Format

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

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
  - ▶ With notes, examples, programs: what will help you on the exam.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ No origami— it's distracting to others taking the exam.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ No origami— it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since excellent way to study.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ No origami— it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since excellent way to study.
- The exam format:

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ No origami— it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
  - ▶ 10 questions, each worth 10 points.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ No origami— it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
  - ▶ 10 questions, each worth 10 points.
  - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ No origami— it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
  - ▶ 10 questions, each worth 10 points.
  - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
  - ▶ Style of questions: what does the code do? short answer, write functions, top down design, & write complete programs.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ No origami— it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
  - ▶ 10 questions, each worth 10 points.
  - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
  - ▶ Style of questions: what does the code do? short answer, write functions, top down design, & write complete programs.
  - ▶ More on logistics next lecture.

# Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ No origami— it's distracting to others taking the exam.
  - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
  - ▶ 10 questions, each worth 10 points.
  - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
  - ▶ Style of questions: what does the code do? short answer, write functions, top down design, & write complete programs.
  - ▶ More on logistics next lecture.
- Past exams available on webpage (includes answer keys).

# Exam Options

## Exam Times:

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

11 December 2018

### Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or commissing, attempting to commit or aiding in the commission of acts of academic dishonesty) as offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy Against Academic Dishonesty and the specific case of academic dishonesty according to the Hunter College Academic Integrity Procedure.

I understand that all cases of academic dishonesty will be reported to the Dean of Students and all records will remain on record.
Name:
English:
Email:
Signature:

# Exam Options

## Exam Times:

- Default Regular Time: Monday, 23 May, 9-11am.

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

18 December 2018

### Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or commissing, assisting or encouraging another to commit acts of academic dishonesty) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy Against Academic Dishonesty. Please review *Code of Academic Dishonesty* according to the Hunter College Academic Integrity Procedure.

I understand that all cases of academic dishonesty will be reported to the Dean of Students and all records will remain on record.
Name: _____
English: _____
Email: _____
Signature: _____

# Exam Options

## Exam Times:

- Default Regular Time: Monday, 23 May, 9-11am.
- Alternate Time: Friday, 20 May, 8am-10am.

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

14 December 2018

### Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pen and pencil, and your note sheet.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or commissing, assisting or encouraging another to commit an act of academic dishonesty) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy Against Academic Dishonesty. Please review *Code of Academic Dishonesty* according to the Hunter College Academic Integrity Procedure.

I understand that all cases of academic dishonesty will be reported to the Dean of Students and all records are permanent.
Name:
EngID:
Email:
Signature:

# Exam Options

## Exam Times:

- Default Regular Time: Monday, 23 May, 9-11am.
- Alternate Time: Friday, 20 May, 8am-10am.
- Accessibility Testing: Paperwork required. Must be completed on 29 April. If you have not done so already, email me no later than 29 April.

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

14 December 2018

### Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or commissing, attempting to commit or aiding in the commission of acts of academic dishonesty) as offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy Against Academic Dishonesty and the Hunter College Academic Integrity Procedure.

I understand that all cases of academic dishonesty will be reported to the Dean of Students and all records will remain.
Name:
EngID:
Email:
Signature:

# Exam Options

## Exam Times:

- Default Regular Time: Monday, 23 May, 9-11am.
- Alternate Time: Friday, 20 May, 8am-10am.
- Accessibility Testing: Paperwork required. Must be completed on 29 April. If you have not done so already, email me no later than 29 April.
- Survey for your exam date choice will be available next lecture. **No survey answer implies you will take the exam on 23 May.**

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

14 December 2018

### Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8.5" x 11" piece of paper that can be used for calculations.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or commencing, attempting, or aiding in the attempt of such acts) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy Against Academic Dishonesty and the Hunter College Academic Integrity Procedure.

<small>Indicate that all cases of academic dishonesty will be reported to the Dean of Students and all grades are舞弊.</small>
Name:
English:
Email:
Signature:

# Exam Options

## Exam Times:

- Default Regular Time: Monday, 23 May, 9-11am.
- Alternate Time: Friday, 20 May, 8am-10am.
- Accessibility Testing: Paperwork required. Must be completed on 29 April. If you have not done so already, email me no later than 29 April.
- Survey for your exam date choice will be available next lecture. **No survey answer implies you will take the exam on 23 May.**
- If you choose to take the early date, **you will not be given access to the exam on 23 May even if you miss the early exam.**

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

14 December 2018

### Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College requires acts of academic dishonesty (e.g., plagiarism, cheating or commissing another person to commit acts of academic dishonesty) to be reported to the Office of Student Affairs upon the review of individual faculty. The College is committed to enforcing the CUNY Policy against Academic Dishonesty. Please see *CUNY Policy Against Academic Dishonesty according to the Hunter College Academic Integrity Procedure*.

<input type="checkbox"/> I understand that all cases of academic dishonesty will be reported to the Office of Student Affairs and all records will remain.
Name: _____
English: _____
Email: _____
Signature: _____

# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab

# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North

# Weekly Reminders!



Before next lecture, don't forget to:

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

# Weekly Reminders!



Before next lecture, don't forget to:

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

# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every week**) in lab 1001G Hunter North
- Submit this week's 5 programming assignments (**programs 51-55**)
- If you need help, schedule an appointment for Tutoring in lab 1001G 11:30am-5pm

# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every week**) in lab 1001G Hunter North
- Submit this week's 5 programming assignments (**programs 51-55**)
- If you need help, schedule an appointment for Tutoring in lab 1001G 11:30am-5pm
- Take the Lecture Preview on Blackboard on Monday (or no later than 10am on Tuesday)

# Lecture Slips & Writing Boards



- Hand your lecture slip to a UTA.
- Return writing boards as you leave.