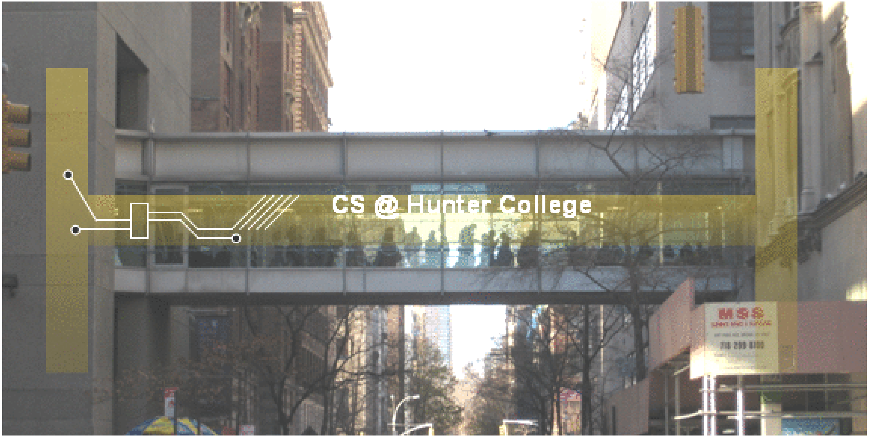
CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Frequently Asked Questions

From email and tutoring.

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

Frequently Asked Questions

From email and tutoring.

- **When is the final? Is there a review sheet?**
  The official final is Monday, December 19, 9-11am.

Frequently Asked Questions

From email and tutoring.

- **When is the final? Is there a review sheet?**
  The official final is Monday, December 19, 9-11am.
  The early final exam (alternative date) is on Friday, December 16, 8-10am.

Frequently Asked Questions

From email and tutoring.

- **When is the final? Is there a review sheet?**
  The official final is Monday, December 19, 9-11am.
  The early final exam (alternative date) is on Friday, December 16, 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, December 19, 9-11am.
  The early final exam (alternative date) is on Friday, December 16, 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, December 19, 9-11am.
  The early final exam (alternative date) is on Friday, December 16, 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, December 19, 9-11am.
  The early final exam (alternative date) is on Friday, December 16, 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 13 December.*

Handle Exam Anxiety – courtesy Dr. St. John

- Print out the past exams and do as much as possible in 1 hour.
- Then grade yourselves, figure out which problems are similar to past problems, keeping all the exams youve done in a 3-hole notebook, 1 problem per page, organized by problem number, reinforces the similarity.
- Make a list of what does not make sense and asking the instructor.
- Attempting to do the exam in half the time means that in the real exam, you will have plenty of time.

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:

```python
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)
```

Predict what the code will do: II

```
nums= [1,4,10,6,5,42,9,8,12]
target = 6
if search(nums, target):
    print(target, ’is in the list.’)
else:
    print(target, ’is not in the list.’)
```

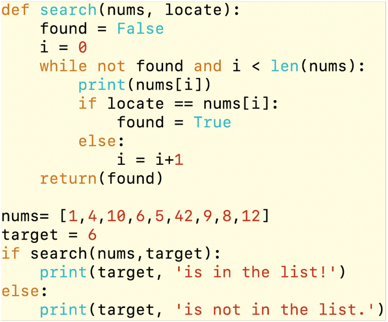
Simplified but a little tricky

```python
def search(nums, locate):
    i = 0
    while i < len(nums) and locate!=nums[i]:
        print(nums[i])
        i = i+1

    return (i < len(nums))
    #If locate is in the list,
    #then for some i < len(nums), we have
    #locate == nums[i].
    #If i >= len(nums), this implies that all
    #items are searched, no match is found.
```

Simplified but a little tricky

```
nums= [1,4,10,6,5,42,9,8,12]
target = 6
if search(nums, target):
    print(target, ’is in the list.’)
else:
    print(target, ’is not in the list.’)
```

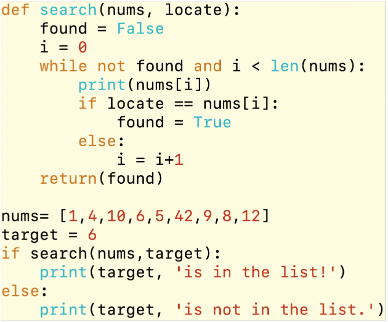
Design Pattern: Linear Search



```python
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]
target = 6
if search(nums,target):
    print(target, 'is in the list!')
else:
    print(target, 'is not in the list.')
```

- Example of **linear search**.

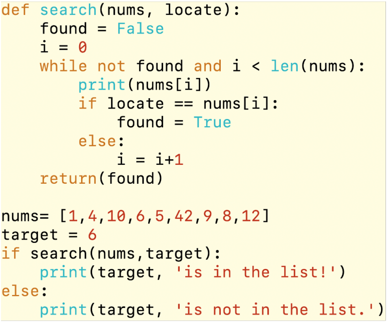
Design Pattern: Linear Search



```
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]
target = 6
if search(nums,target):
    print(target, 'is in the list!')
else:
    print(target, 'is not in the list.')
```

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

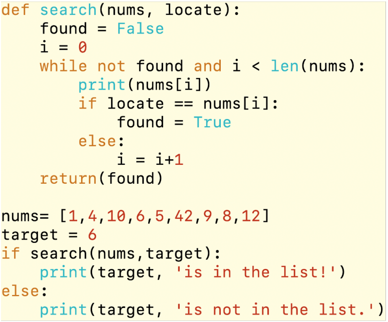
Design Pattern: Linear Search



```
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]
target = 6
if search(nums,target):
    print(target, 'is in the list!')
else:
    print(target, 'is not 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]
target = 6
if search(nums,target):
    print(target, 'is in the list!')
else:
    print(target, 'is not 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 lis 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

```python
#Texts following # are comments.
#Comments are read by human beings, not computer.
#Name: Thomas Hunter
#Date: September 1, 2017
#This program prints: Hello, World!

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

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

```python
import turtle

taylor = turtle.Turtle()
taylor.color("purple")
taylor.shape("turtle")

n = 6
for i in range(n):
    taylor.forward(100)
    taylor.stamp()
    taylor.left(360/n)
```

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.

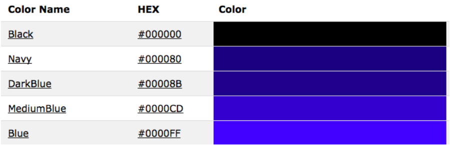
Week 2: variables, data types, more on loops & range()

- 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:

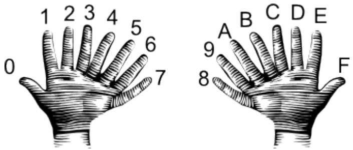
Examples on loop and ranges

```
for num in [2,4,6,8,10]:
    print(num)

sum = 0
for x in range(0,12,2):
    print(x)
    sum += x

print(sum)

for c in 'ABCD':
    print(c)
```

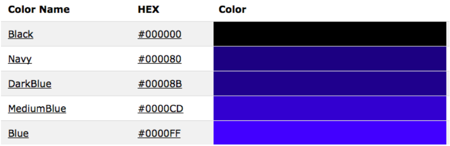
Week 3: colors, hex, slices, numpy & images



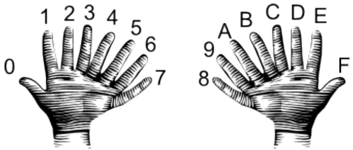
Color Name	HEX	Color
Black	#000000	⬛
Navy	#000080	🟦
DarkBlue	#00008B	🟦
MediumBlue	#0000CD	🟦
Blue	#0000FF	🟦

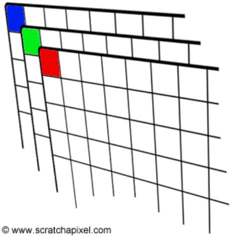


Week 3: colors, hex, slices, numpy & images



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





© www.scratchapixel.com

Two Dimensional Array Slicing

```python
import numpy as np

numRows = 6
numCols = 6
a = np.zeros((numRows, numCols))
#create a table with 6 rows and 6 columns,
#each element is initialized to be zero.
#Do not forget parentheses around
#numRows, numCols.
```

Two Dimensional Array Slicing: II

```
for i in range(numRows):
    for j in range(numCols):
        a[i, j] = i*10 + j
#range(numRows) returns [0, 1, 2, 3, 4, 5],
#where outer loop variable i chooses from.
#When i is 0, run
#    for j in range(numCols):
#        a[i, j] = i*10 + j
#When i is 1, run
#    for j in range(numCols):
#        a[i, j] = i*10 + j
#The last round of i is 5.
```

Two Dimensional Array Slicing: III

```python
for i in range(numRows):
    for j in range(numCols):
        print("%3i"%(a[i, j]), end="")
        #"%3i"%(a[i, j]) prints a[i, j] --
        #element of a at ith row and
        #jth column -- as an 3-digit int.
        #"%3i" is a place holder and is filled by a[i, j].
        #If a[i, j] does not have 3 digits,
        #pad space(s) to the left.
        #end="" print w/o a new line.

    print() #print a new line after each row
```

Two Dimensional Array Slicing: III

```
print(a[0, 3:5])
```

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

Two Dimensional Array Slicing: III

```
print(a[0, 3:5])
```

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

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

print

```
[3. 4.]
```

Two Dimensional Array Slicing: IV

```python
print(a[4:, 4:])
```

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

Two Dimensional Array Slicing: IV

```
print(a[4:, 4:])
```

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

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

Print out

```
[[44. 45.]
 [54. 55.]]
```

Two Dimensional Array Slicing: V

```python
print(a[:, 2])
```

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

Two Dimensional Array Slicing: V

```
print(a[:, 2])
```

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

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

Print out

```
[ 2. 12. 22. 32. 42. 52.]
```

Two Dimensional Array Slicing: VI

```python
print(a[2::2, ::2])
```

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

Two Dimensional Array Slicing: VI

```python
print(a[2::2, ::2])
```

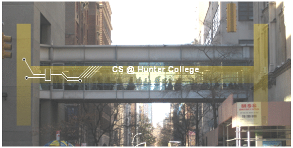
	0	1	2	3	4	5
0	0	1	2	3	4	5
1	10	11	12	13	14	15
2	20	21	22	23	24	25
3	30	31	32	33	34	35
4	40	41	42	43	44	45
5	50	51	52	53	54	55

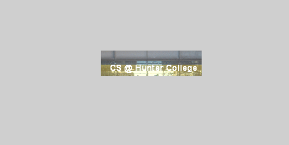
	0	1	2	3	4	5
0	0	1	2	3	4	5
1	10	11	12	13	14	15
2	20	21	22	23	24	25
3	30	31	32	33	34	35
4	40	41	42	43	44	45
5	50	51	52	53	54	55

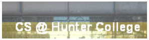
print

```
[[20. 22. 24.]
 [40. 42. 44.]]
```

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.

Grayed surrounding area of an image

```python
#Leave middle 1/5 height * 1/2 width unchanged,
#gray the rest area of the image
import matplotlib.pyplot as plt
import numpy as np

fileName = input("Enter a file name: ")
img = plt.imread(fileName)

height = img.shape[0]
width = img.shape[1]
```

Grayed surrounding area of an image: II

```
#make the top 2/5 area gray
img[:height*2//5, :] = [0.5, 0.5, 0.5, 1]
#[0.5, 0.5, 0.5, 1] means
#red 0.5, green 0.5, blue 0.5 and opacity 1

#make the bottom 1/4 area gray
img[-height*2//5:, :] = [0.5, 0.5, 0.5, 1]
#img[-height*2//5:, :] same as
#img[height*3//5:, :]
##height*2//5 from BOTTOM sams as
#height*3//5 from TOP
#img[height*3//5:, :] = [0.5, 0.5, 0.5, 1]
```

Grayed surrounding area of an image: III

```
#make the left 1/4 area gray
img[:, :width//4] = [0.5, 0.5, 0.5, 1]

#make the right 1/4 area gray
img[:, width*3//4:] = [0.5, 0.5, 0.5, 1]
#img[:, width*3//4: ] same as
#img[:, -width//4: ]
#width*3//4 from LEFT same as
#width//4 from RIGHT
#img[:, -width//4:] = [0.5, 0.5, 0.5, 1]

plt.imshow(img)
plt.show()
```

Highlight part of image

```python
#Leave middle 1/5 height * 1/2 width
#section unchanged,
#dim the rest area of the image
#It is like to highlight the middle part.

import matplotlib.pyplot as plt
import numpy as np

fileName = "csBridge.png"
#fileName = input("Enter a file name: ")
img = plt.imread(fileName)
```

Highlight part of image: II

```
height = img.shape[0]
width = img.shape[1]

#make the top 1/4 area gray
#img[:height*2//5, :] = [0.5, 0.5, 0.5, 1]
#[0.5, 0.5, 0.5, 1] means
#red 0.5, green 0.5, blue 0.5 and opacity 1
#unlike user created stripe images,
#some images have four channels.

#dim the top 1/4 area
img[:height*2//5, :, 3] = 0.5
```

Highlight part of image: III

```
#dim the bottom 1/4 area
img[-height*2//5:, :, 3] = 0.5
#img[-height*2//5:, :, 3] same as img[height*3//5:, :, 3]
#height*2//5 from BOTTOM same as height*3//5 from TOP

#dim the left 1/4 area
img[:, :width//4, 3] = 0.5
```

Highlight part of image: IV

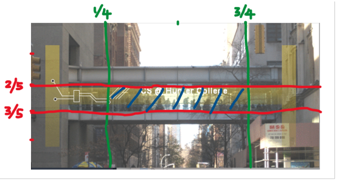
```
#dim the right 1/4 area
img[:, width*3//4:, 3] = 0.5
#img[:, width*3//4:, 3] same as
#img[:, -width//4:, 3]
#width*3//4 from LEFT same as
#width//4 from RIGHT

plt.imshow(img)
plt.show()
```

Crop image

```python
#crop middle 1/5 * height * 1/2 * width area
import matplotlib.pyplot as plt
import numpy as np

fileName = "csBridge.png"
#fileName = input("Enter a file name: ")
img = plt.imread(fileName)

height = img.shape[0]
width = img.shape[1]
```

Crop image: II

```
img2 = img[height*2//5 : height*3//5, width
    *1//4 : width*3//4] #// cannot be replaced
     by /, indices need to be int.

plt.imshow(img2)
plt.show()

plt.imsave('cropped_image.png', img2)
```



Week 4: design problem (cropping images) & decisions

```python
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")
```

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

```
x = int(input("Enter number: "))

if x % 2 == 0:
    print("Even number")
else:
    print("Odd number")
```

Week 5: logical operators, truth tables & logical circuits

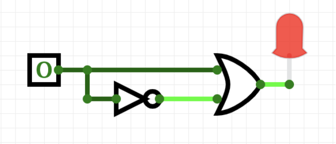
```
origin = "Indian Ocean"
winds = 100
if winds >= 74:
    print("Major storms, called a ", end="")
    if origin == "Indian Ocean" or origin == "South Pacific":
        print("cyclone.")
    elif origin == "North Pacific":
        print("typhoon.")
    else:
        print("hurricane.")
```

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

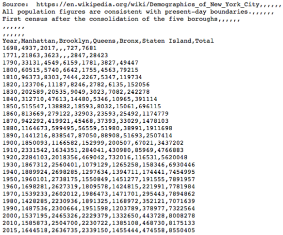
```python
visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and
    (conditions == "blowing snow" or
      conditions == "heavy snow"):
    print("Blizzard!")
```

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

in1		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



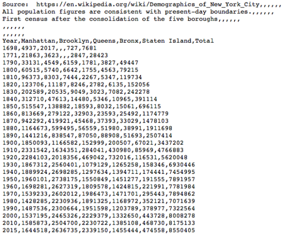
```
Source:  https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,,
All population figures are consistent with present-day boundaries.,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,,
,,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6642,1755,4563,79215
1810,96373,8303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,5346,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813669,279122,32903,23593,25492,1174779
1870,942292,419921,45468,37393,33029,1478103
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,152999,200507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018356,469042,732016,116531,5620048
1930,1867312,2560401,1079129,1265258,158346,6930446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550849,1451277,191555,7891957
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1585873,2504700,2230722,1385108,468730,8175133
2015,1644518,2636735,2339150,1455444,474558,8550405
```

`nycHistPop.csv`

In Lab 6

Week 6: structured data, pandas, & more design

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



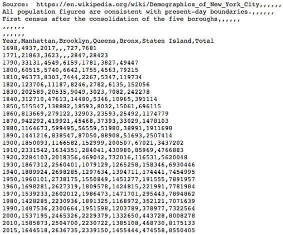
```
Source:  https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,,
All population figures are consistent with present-day boundaries.,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,,
,,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6642,1755,4563,79215
1810,96373,8303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,5346,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813669,279122,32903,23593,25492,1174779
1870,942292,419921,45468,37393,33029,1478103
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,152999,200507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018356,469042,732016,116531,5620048
1930,1867312,2560401,1079129,1265258,158346,6930446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550849,1451277,191555,7891957
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1585873,2504700,2230722,1385108,468730,8175133
2015,1644518,2636735,2339150,1455444,474558,8550405
```

`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)
```



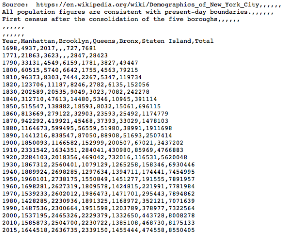
```
Source:  https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,,
All population figures are consistent with present-day boundaries.,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,,
,,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6642,1755,4563,79215
1810,96373,8303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,5346,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813669,279122,32903,23593,25492,1174779
1870,942292,419921,45468,37393,33029,1478103
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,152999,200507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018356,469042,732016,116531,5620048
1930,1867312,2560401,1079129,1265258,158346,6930446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550849,1451277,191555,7891957
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1585873,2504700,2230722,1385108,468730,8175133
2015,1644518,2636735,2339150,1455444,474558,8550405
```

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)

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



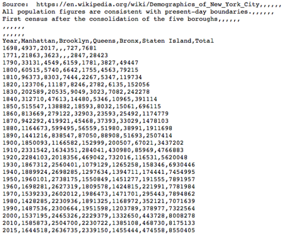
```
Source:  https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,,
All population figures are consistent with present-day boundaries.,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,,
,,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6642,1755,4563,79215
1810,96373,8303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,5346,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813669,279122,32903,23593,25492,1174779
1870,942292,419921,45468,37393,33029,1478103
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,152999,200507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018356,469042,732016,116531,5620048
1930,1867312,2560401,1079129,1265258,158346,6930446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550849,1451277,191555,7891957
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1585873,2504700,2230722,1385108,468730,8175133
2015,1644518,2636735,2339150,1455444,474558,8550405
```

nycHistPop.csv

In Lab 6

Week 6: structured data, pandas, & more design

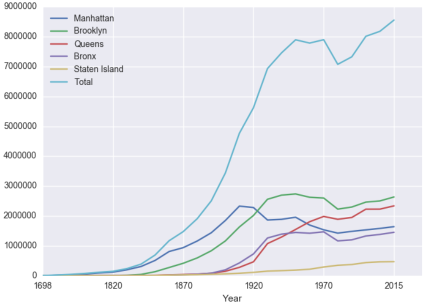
```
import matplotlib.pyplot as plt
import pandas as pd

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

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



```
Source:  https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,,
All population figures are consistent with present-day boundaries.,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,,
,,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6642,1755,4563,79215
1810,96373,8303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,5346,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813669,279122,32903,23593,25492,1174779
1870,942292,419921,45468,37393,33029,1478103
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,152999,200507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018356,469042,732016,116531,5620048
1930,1867312,2560401,1079129,1265258,158346,6930446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550849,1451277,191555,7891957
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1585873,2504700,2230722,1385108,468730,8175133
2015,1644518,2636735,2339150,1455444,474558,8550405
```

`nycHistPop.csv`

In Lab 6



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.

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()
```

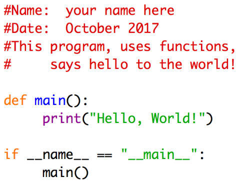
- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`

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:

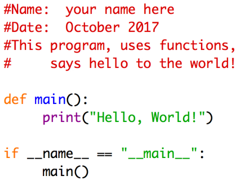
Week 7: functions



```
#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")`

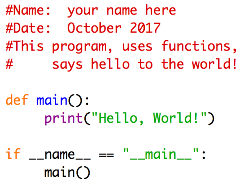
Week 7: functions



```
#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

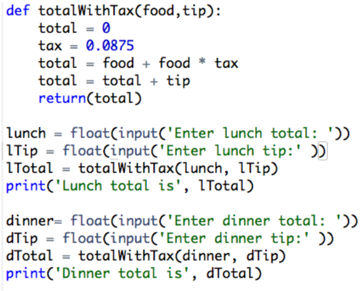


```
#Name:  your name here
#Date:  October 2017
#This program, uses functions,
#     says hello to the world!

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.

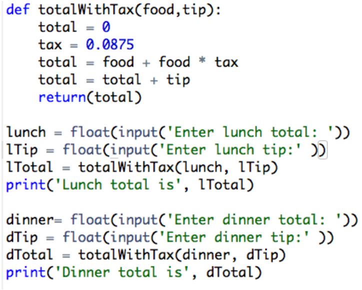
Week 8: function parameters, github

- 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)
```

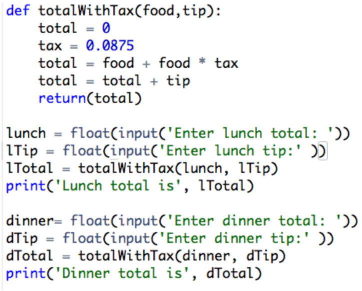
Week 8: function parameters, github



```
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).

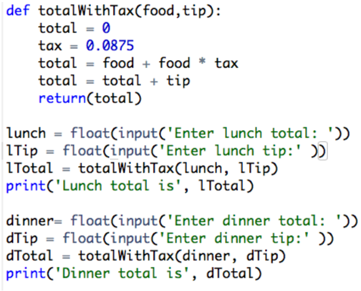
Week 8: function parameters, github



```
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**.

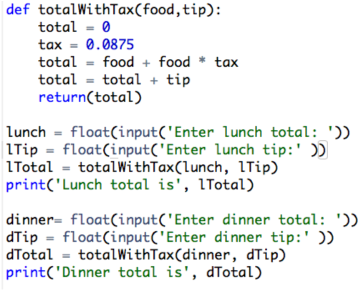
Week 8: function parameters, github



```
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**

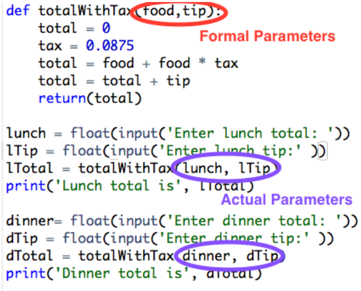
Week 8: function parameters, github



```
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

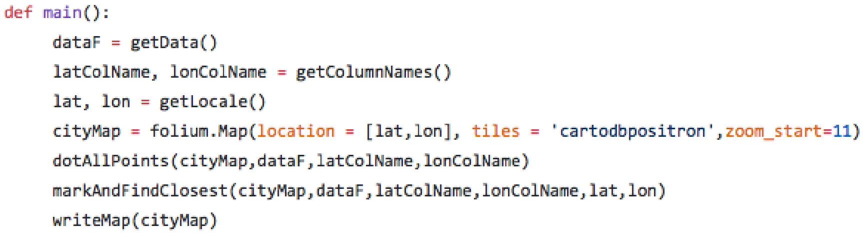


```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

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

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.

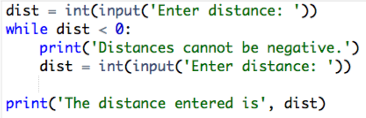
Week 9: top-down design, folium, loops, and random()

+
−
EEHAWKEN
Hudson River
Hell Gate
East River
BROADWAY



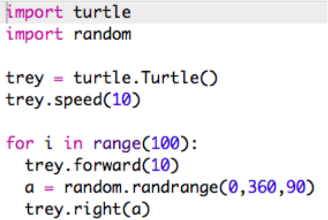
```
def main():
    dataF = getData()
    latColName, lonColName = getColumnNames()
    lat, lon = getLocale()
    cityMap = folium.Map(location = [lat,lon], tiles = 'cartodbpositron',zoom_start=11)
    dotAllPoints(cityMap,dataF,latColName,lonColName)
    markAndFindClosest(cityMap,dataF,latColName,lonColName,lat,lon)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

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



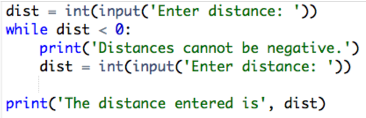
```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

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

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



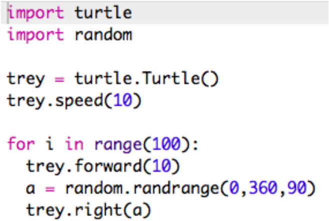
```
import turtle
import random

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()



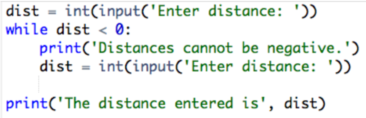
```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

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.

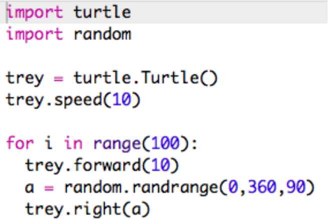


```
import turtle
import random

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()



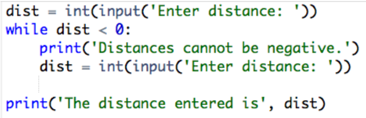
```python
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

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



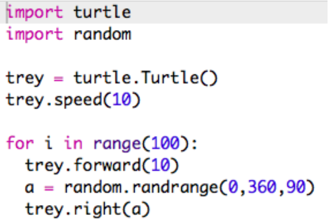
```python
import turtle
import random

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()



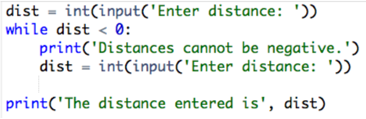
```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

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



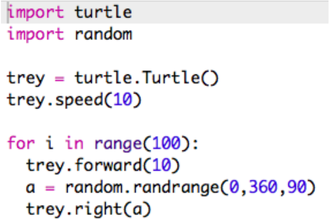
```
import turtle
import random

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.

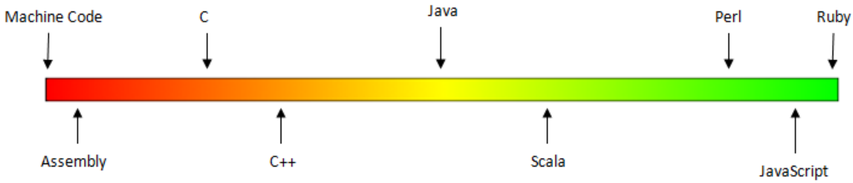
Python & Circuits Review: 10 Weeks in 10 Minutes

- 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

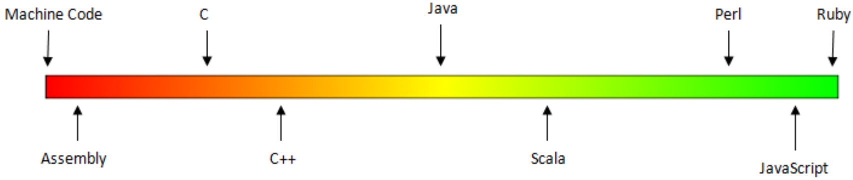
Low-Level vs. High-Level Languages



(codeCommit)

- Can view programming languages on a continuum.

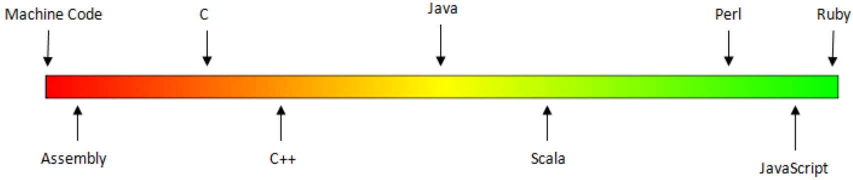
Low-Level vs. High-Level Languages



(codeCommit)

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

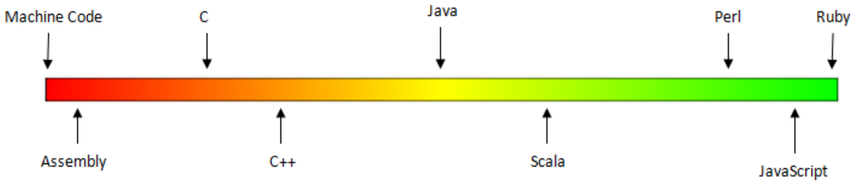
Low-Level vs. High-Level Languages



(codeCommit)

- 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).

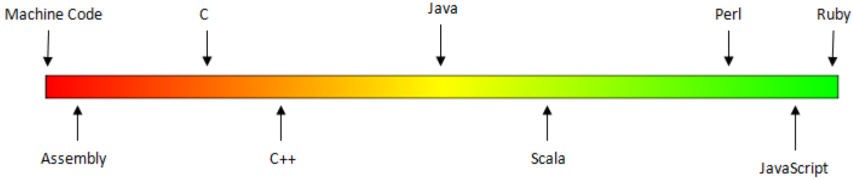
Low-Level vs. High-Level Languages



(codeCommit)

- 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

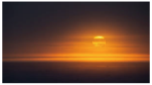


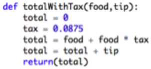
(codeCommit)

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

Processing

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

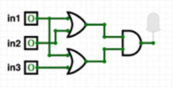




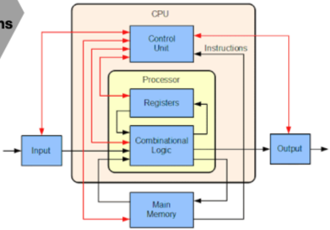
```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)
```

Data & Instructions

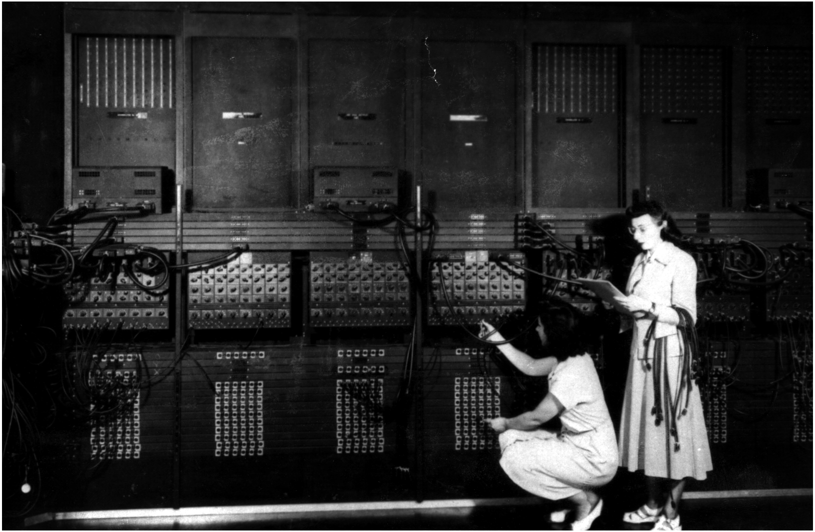
Data & Instructions



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

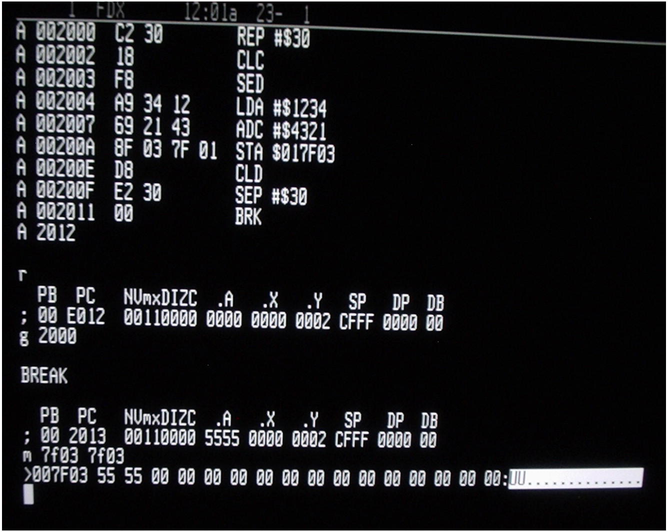


Machine Language



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

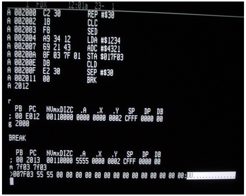
Machine Language



```
      I  FDX        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   NVmxDIZC  .A   .X   .Y   SP   DP  DB
; 00 E012  00110000 0000 0000 0002 CFFF 0000 00
g 2000

BREAK

  PB  PC   NVmxDIZC  .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 00:UU..............
```

(wiki)

Machine Language

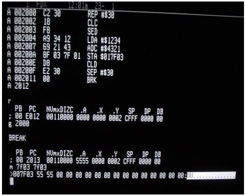


```
      1  FDX       12:01a  23-  1
A 002000  C2 30          REP #$30
A 002002  18             CLC
A 002003  F8             SED
A 002004  A9 34 12       LDA #$1234
A 002007  69 21 43       ADC #$4321
A 00200A  8F 03 7F 01    STA $017F03
A 00200E  D8             CLD
A 00200F  E2 30          SEP #$30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

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

BREAK

  PB  PC   NVmxDIZC  .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 00:UU..............
```

(wiki)

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

Machine Language

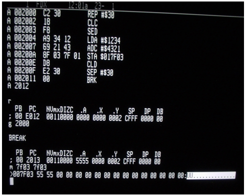


```
     1  FDX       12:01a  23-  1
A 002000  C2 30        REP #$30
A 002002  18           CLC
A 002003  F8           SED
A 002004  A9 34 12     LDA #$1234
A 002007  69 21 43     ADC #$4321
A 00200A  8F 03 7F 01  STA $017F03
A 00200E  D8           CLD
A 00200F  E2 30        SEP #$30
A 002011  00   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

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

BREAK

  PB  PC   NVmxDIZC  .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 00:UU..............
```

(wiki)

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

Machine Language

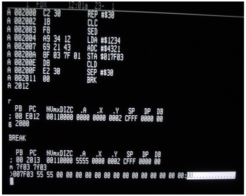


```
     1  FDX       12:01a  23-  1
A 002000  C2 30         REP #$30
A 002002  18            CLC
A 002003  F8            SED
A 002004  A9 34 12      LDA #$1234
A 002007  69 21 43      ADC #$4321
A 00200A  8F 03 7F 01   STA $017F03
A 00200E  D8            CLD
A 00200F  E2 30         SEP #$30
A 002011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

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

BREAK

  PB  PC   NVmxDIZC  .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 00:UU..............
```

(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



```
     1  FDX       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   NVmxDIZC  .A   .X   .Y   SP   DP  DB
; 00 E012  00110000 0000 0000 0002 CFFF 0000 00
g 2000

BREAK

  PB  PC   NVmxDIZC  .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 00:UU..............
```

(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

```
# Store ’Hello world!’ at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # 72 is ASCII code of ’H’
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
```

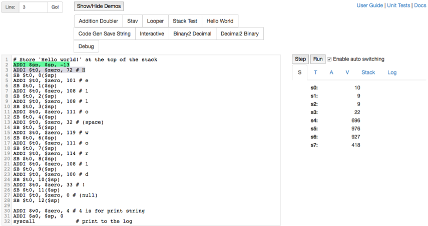
“Hello World!” in Simplified Machine Language: II

```
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
```

"Hello World!" in Simplified Machine Language: II

```
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
 syscall             # print to the log
```

WeMIPS

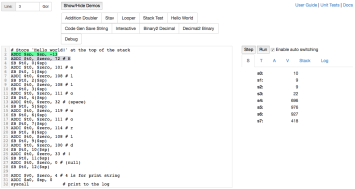


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall             # print to the log
```

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

(Demo with `WeMIPS`)

MIPS Commands

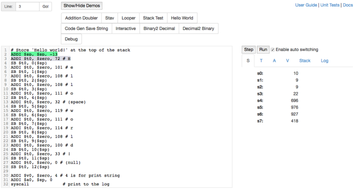


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall              # print to the log
```

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

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

MIPS Commands

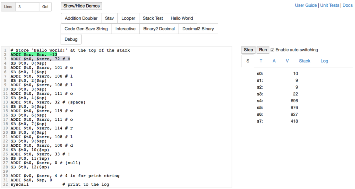


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall            # print to the log
```

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

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

MIPS Commands



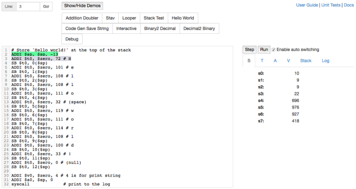
Line: 3

```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall             # print to the log
```

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



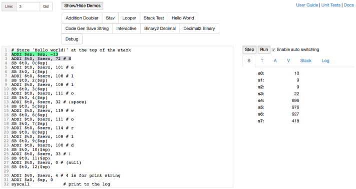
Line: 3 Go!

```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall             # print to the log
```

```
s0:   10
s1:    9
s2:    9
s3:   22
s4:  696
s5:  976
s6:  927
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:
  `add $s1, $s2, $s3`

MIPS Commands

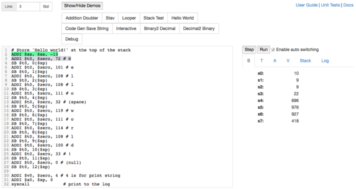


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall             # print to the log
```

s0:	10
s1:	9
s2:	9
s3:	22
s4:	696
s5:	976
s6:	927
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: `add $s1, $s2, $s3` (Basic form: `OP rd, rs, rt`)
- **I Instructions:** instructions that also use intermediate values.

MIPS Commands



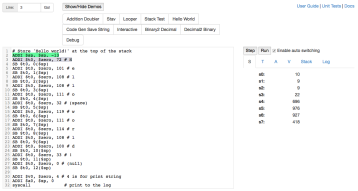
Line: 3 Go!
Show/Hide Demos
User Guide | Unit Tests | Docs
Addition Doubler | Stav | Looper | Stack Test | Hello World
Code Gen Save String | Interactive | Binary2 Decimal | Decimal2 Binary
Debug
Step | Run | ☑ Enable auto switching
S T A V Stack Log

```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall             # print to the log
```

s0:	10
s1:	9
s2:	9
s3:	22
s4:	696
s5:	976
s6:	927
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:
  `add $s1, $s2, $s3`   (Basic form: `OP rd, rs, rt`)
- **I Instructions:** instructions that also use intermediate values.
  `addi $s1, $s2, 100`

MIPS Commands

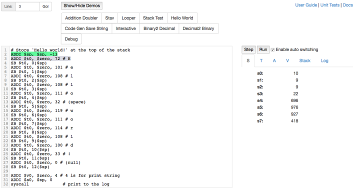


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall             # print to the log
```

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

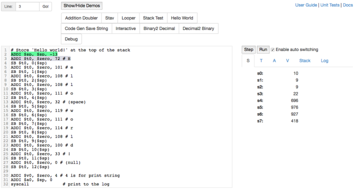


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall             # print to the log
```

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

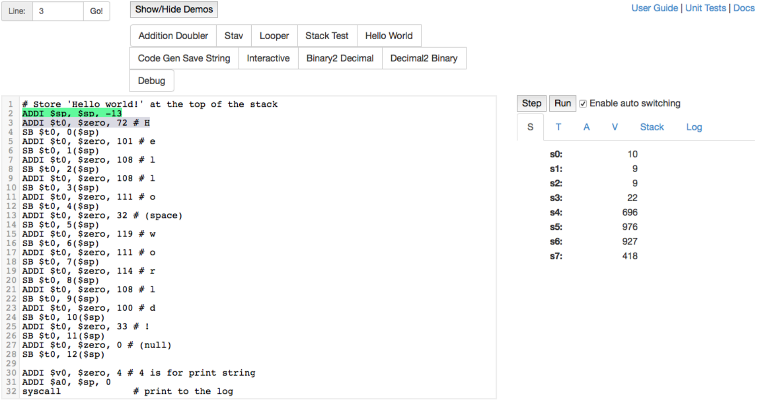


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall            # print to the log
```

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

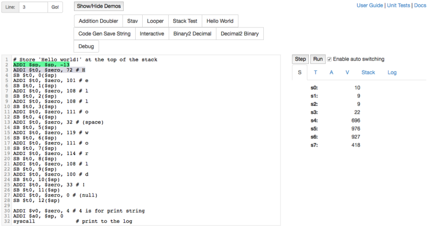


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall            # print to the 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



```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall             # print to the log
```

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

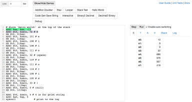
(Demo with `WeMIPS`)

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.

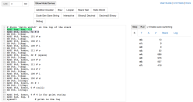


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall          # print to the log
```

```
s0:   10
s1:    8
s2:    9
s3:   22
s4:  686
s5:  976
s6:  927
s7:  418
```

Loops & Jumps in Machine Language

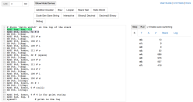
- Instead of built-in looping structures like `for` and `while`, you create your own loops by "jumping" to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.



```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall              # print to the log
```

s0:	10
s1:	9
s2:	9
s3:	22
s4:	686
s5:	976
s6:	927
s7:	418

Loops & Jumps in Machine Language

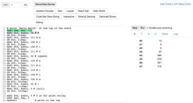


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall            # print to the log
```

```
s0:    10
s1:     8
s2:     9
s3:    22
s4:   686
s5:   976
s6:   927
s7:   418
```

- 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

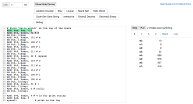


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall            # print to the log
```

```
s0:    10
s1:     9
s2:     9
s3:    22
s4:   686
s5:   976
s6:   927
s7:   418
```

- 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

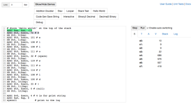


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall            # print to the log
```

s0:	10
s1:	9
s2:	9
s3:	22
s4:	686
s5:	976
s6:	927
s7:	418

- 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

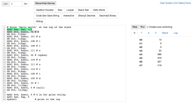


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall            # print to the log
```

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

- 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



```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall            # print to the log
```

```
s0:   10
s1:    8
s2:    9
s3:   22
s4:  686
s5:  976
s6:  927
s7:  418
```

- 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.

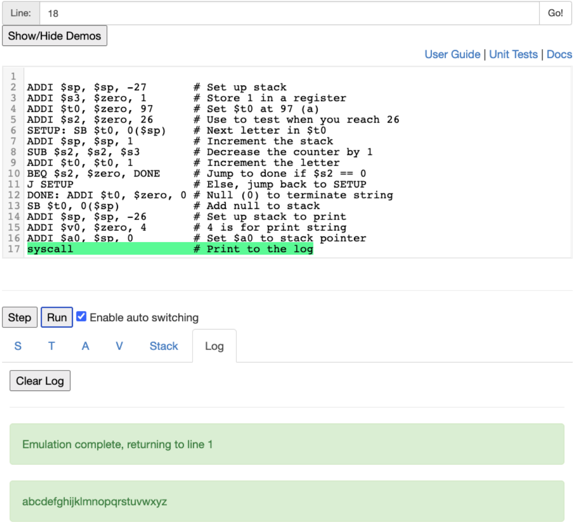
Print alphabet table in Simplified Machine Language: II

```
ADDI $sp, $sp, -27 #setup stack, 26 letters + 1 null
ADDI $t0, $zero, 97 #save ASCII of 'a' to $t0
ADDI $s2, $zero, 26 #set $s2 to be 26, track whether 26 is reached or not
SETUP:SB $t0, 0($sp) #save contents of $t0 to stack
ADDI $sp, $sp, 1 #increment the stack
ADDI $s2, $s2, -1 #subtract 1 from $s2
ADDI $t0, $t0, 1 #increment the letter
BEQ $s2, $zero, DONE
J SETUP
```

Print alphabet table in Simplified Machine Language: II

```
DONE:ADDI $t0, $zero, 0 #set null
SB $t0, 0($sp)
ADDI $sp, $sp, -26
ADDI $v0, $zero, 4 #$v0 is 4 means to print
ADDI $a0, $sp, 0 #set $a0 to stack pointer
 syscall
```

Jump Demo



```

ADDI $sp, $sp, -27        # Set up stack
ADDI $s3, $zero, 1        # Store 1 in a register
ADDI $t0, $zero, 97       # Set $t0 at 97 (a)
ADDI $s2, $zero, 26       # Use to test when you reach 26
SETUP: SB $t0, 0($sp)     # Next letter in $t0
ADDI $sp, $sp, 1          # Increment the stack
SUB $s2, $s2, $s3         # Decrease the counter by 1
ADDI $t0, $t0, 1          # Increment the letter
BEQ $s2, $zero, DONE      # Jump to done if $s2 == 0
J SETUP                   # Else, jump back to SETUP
DONE: ADDI $t0, $zero, 0 # Null (0) to terminate string
SB $t0, 0($sp)            # Add null to stack
ADDI $sp, $sp, -26        # Set up stack to print
ADDI $v0, $zero, 4        # 4 is for print string
ADDI $a0, $sp, 0          # Set $a0 to stack pointer
syscall                   # Print to the log
```

```
Emulation complete, returning to line 1
```

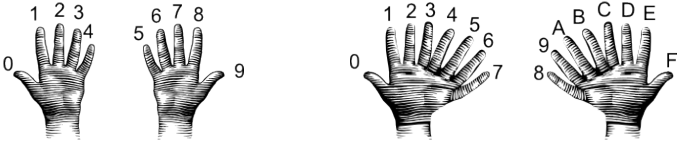
```
abcdefghijklmnopqrstuvwxyz
```

(Demo with WeMIPS)

Today's Topics

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

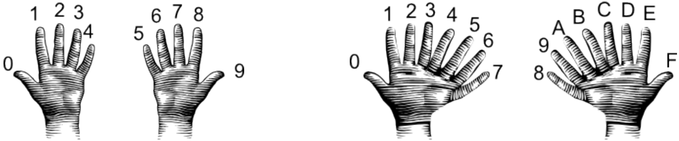
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

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

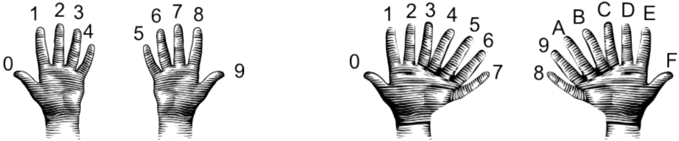
Hexadecimal to Decimal: Converting Between Bases



(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.

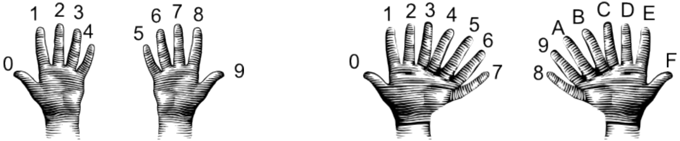
Hexadecimal to Decimal: Converting Between Bases



(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

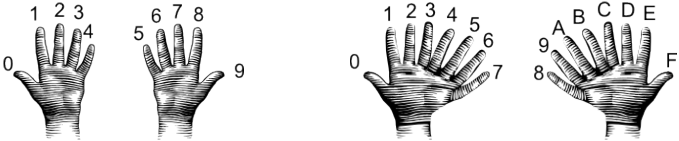


(from i-programmer.info)

- 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

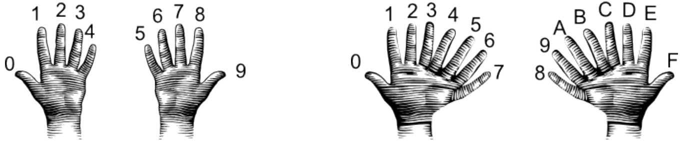


(from i-programmer.info)

- 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*16 is 32.`

Hexadecimal to Decimal: Converting Between Bases

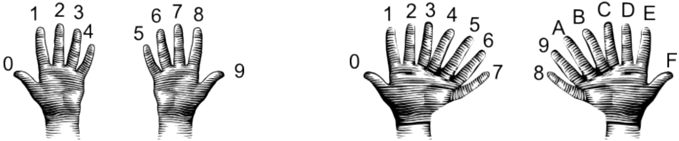


(from i-programmer.info)

- 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*16 is 32.
A in decimal digits is 10.
```

Hexadecimal to Decimal: Converting Between Bases

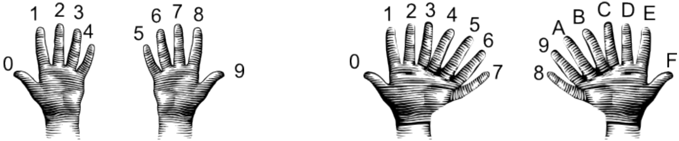


(from i-programmer.info)

- 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*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?

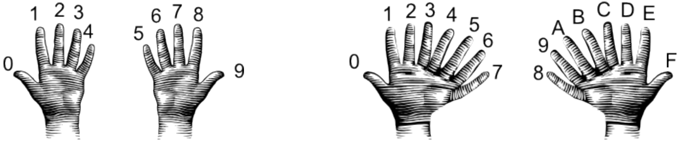
    `2 in decimal is 2. 2*16 is 32.`

    `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*16 is 32.`

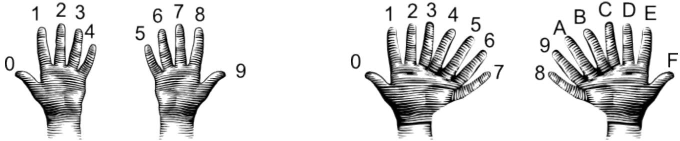
    `A in decimal digits is 10.`

    `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*16 is 32.`

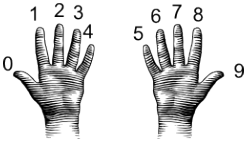
    `A in decimal digits is 10.`

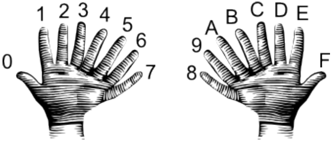
    `32 + 10 is 42.`

    Answer is 42.
  - Example: what is 99 as a decimal number?

    `9 in decimal is 9. 9*16 is 144.`

Hexadecimal to Decimal: Converting Between Bases





(from i-programmer.info)

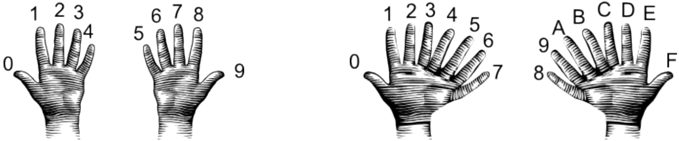
- From hexadecimal to decimal (assuming two-digit numbers):
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?

    ```
    2 in decimal is 2. 2*16 is 32.
    A in decimal digits is 10.
    32 + 10 is 42.
    ```

    Answer is 42.
  - Example: what is 99 as a decimal number?

    ```
    9 in decimal is 9. 9*16 is 144.
    9 in decimal digits is 9
    ```

Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (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*16 is 32.`

    `A in decimal digits is 10.`

    `32 + 10 is 42.`

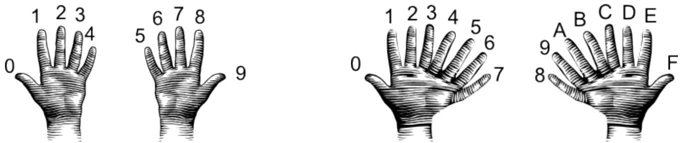
    Answer is 42.
  - Example: what is 99 as a decimal number?

    `9 in decimal is 9. 9*16 is 144.`

    `9 in decimal digits is 9`

    `144 + 9 is 153.`

Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (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*16 is 32.`

    `A in decimal digits is 10.`

    `32 + 10 is 42.`

    Answer is 42.
  - Example: what is 99 as a decimal number?

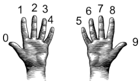
    `9 in decimal is 9. 9*16 is 144.`

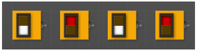
    `9 in decimal digits is 9`

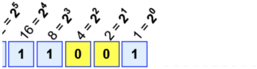
    `144 + 9 is 153.`

    Answer is 153.

Decimal to Binary: Converting Between Bases



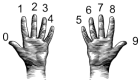


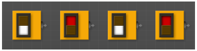


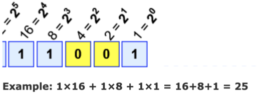
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

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

Decimal to Binary: Converting Between Bases



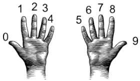


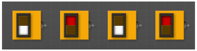


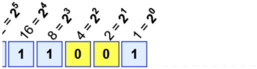
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

- 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



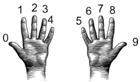


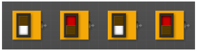


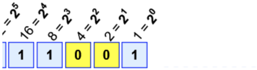
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

- 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



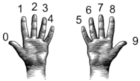


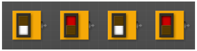


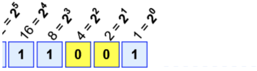
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

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

Decimal to Binary: Converting Between Bases



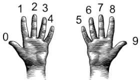


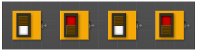


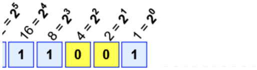
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

- 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.

Decimal to Binary: Converting Between Bases



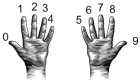


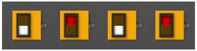


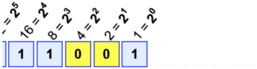
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

- 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



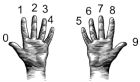


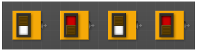


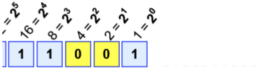
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

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

Decimal to Binary: Converting Between Bases



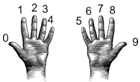


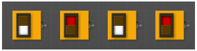


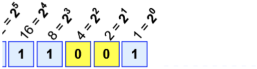
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

- 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



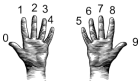


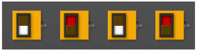


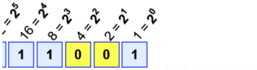
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

- 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





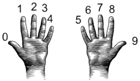


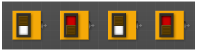
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

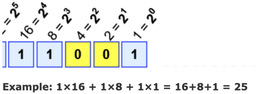
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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



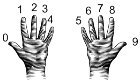


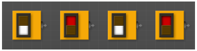


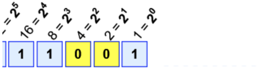
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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





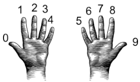


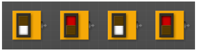
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

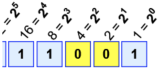
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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

```
130/128 is 1 rem 2. First digit is 1:   1...
2/64 is 0 rem 2.
```

Decimal to Binary: Converting Between Bases





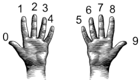


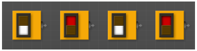
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

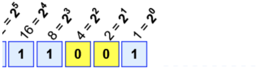
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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





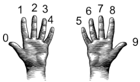


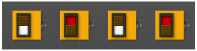
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

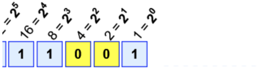
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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





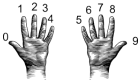


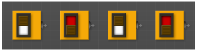
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

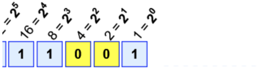
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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





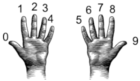


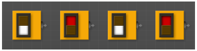
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

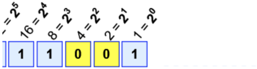
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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





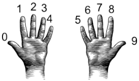


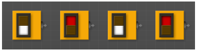
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

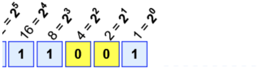
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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





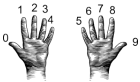


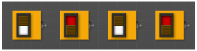
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

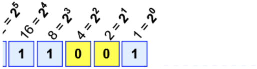
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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

Decimal to Binary: Converting Between Bases





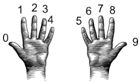


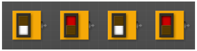
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

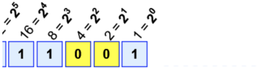
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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





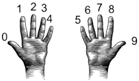


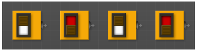
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

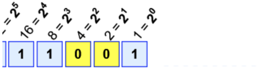
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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

```
130/128 is 1 rem 2. First digit is 1:    1...
2/64 is 0 rem 2. Next digit is 0:        10...
2/32 is 0 rem 2. Next digit is 0:        100...
2/16 is 0 rem 2. Next digit is 0:        1000...
```

Decimal to Binary: Converting Between Bases





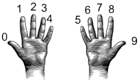


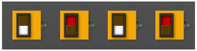
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

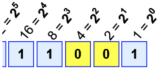
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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

```
130/128 is 1 rem 2. First digit is 1:   1...
2/64 is 0 rem 2. Next digit is 0:       10...
2/32 is 0 rem 2. Next digit is 0:       100...
2/16 is 0 rem 2. Next digit is 0:       1000...
2/8 is 0 rem 2.
```

Decimal to Binary: Converting Between Bases





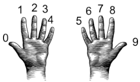


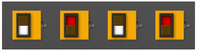
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

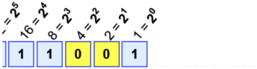
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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





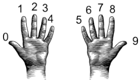


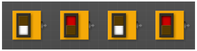
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

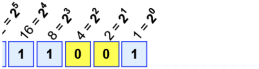
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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





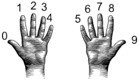


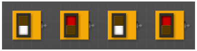
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

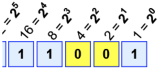
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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

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

Decimal to Binary: Converting Between Bases





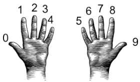


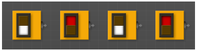
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

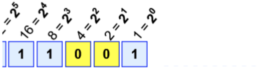
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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





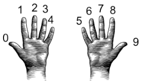


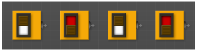
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

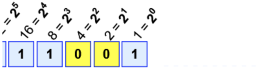
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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

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

Decimal to Binary: Converting Between Bases





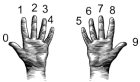


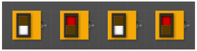
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

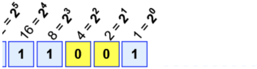
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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





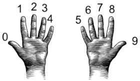


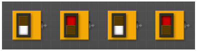
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

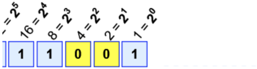
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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





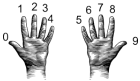


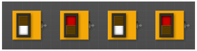
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

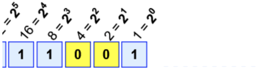
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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

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

Decimal to Binary: Converting Between Bases





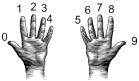


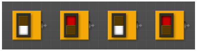
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

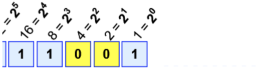
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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





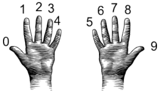


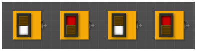
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

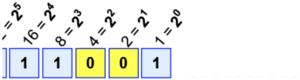
- From decimal to binary:
  - Divide by 128 ($= 2^7$). Quotient is the first digit.
  - Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
  - Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
  - Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
  - Divide remainder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



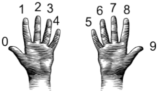


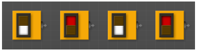


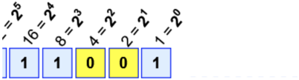
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

- Example: what is 99 in binary notation?

Decimal to Binary: Converting Between Bases





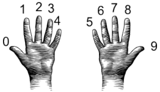


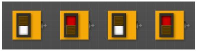
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

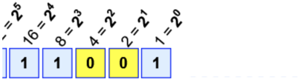
- Example: what is 99 in binary notation?

```
99/128 is 0 rem 99.
```

Decimal to Binary: Converting Between Bases





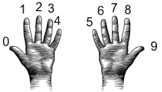


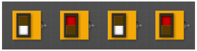
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

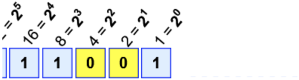
- Example: what is 99 in binary notation?

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

Decimal to Binary: Converting Between Bases





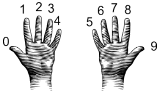


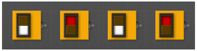
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

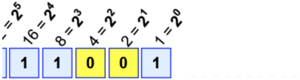
- Example: what is 99 in binary notation?

```
99/128 is 0 rem 99. First digit is 0:    0...
99/64 is 1 rem 35.
```

Decimal to Binary: Converting Between Bases





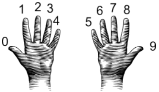


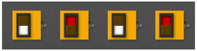
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

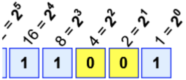
- Example: what is 99 in binary notation?

```
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





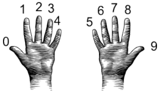


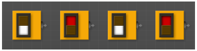
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

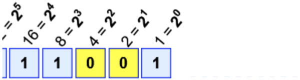
- Example: what is 99 in binary notation?

```
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





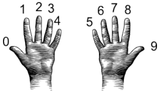


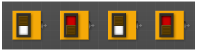
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

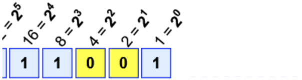
- Example: what is 99 in binary notation?

```
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





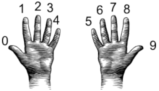


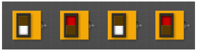
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

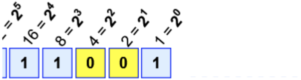
- Example: what is 99 in binary notation?

```
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





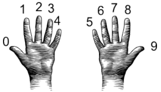


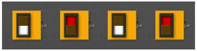
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

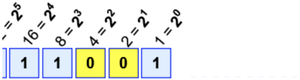
- Example: what is 99 in binary notation?

```
99/128 is 0 rem 99. First digit is 0:    0...
99/64 is 1 rem 35. Next digit is 1:      01...
35/32 is 1 rem 3. Next digit is 1:       011...
```

Decimal to Binary: Converting Between Bases







Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

- Example: what is 99 in binary notation?

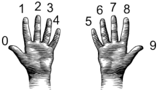
  99/128 is 0 rem 99. First digit is 0: 0...

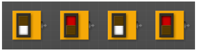
  99/64 is 1 rem 35. Next digit is 1: 01...

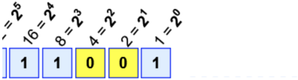
  35/32 is 1 rem 3. Next digit is 1: 011...

  3/16 is 0 rem 3.

Decimal to Binary: Converting Between Bases





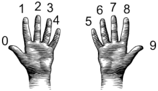


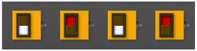
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

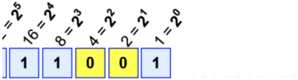
- Example: what is 99 in binary notation?

```
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





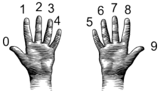


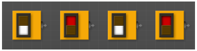
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

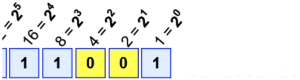
- Example: what is 99 in binary notation?

```
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





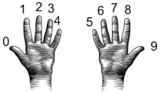


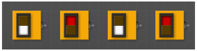
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

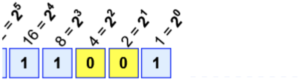
- Example: what is 99 in binary notation?

```
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





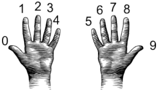


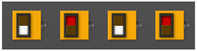
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

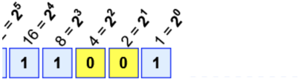
- Example: what is 99 in binary notation?

```
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





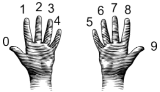


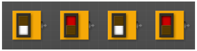
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

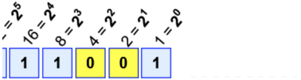
- Example: what is 99 in binary notation?

```
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





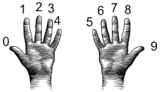


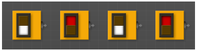
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

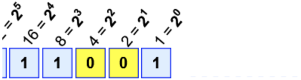
- Example: what is 99 in binary notation?

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

Decimal to Binary: Converting Between Bases





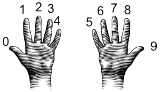


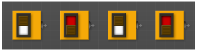
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

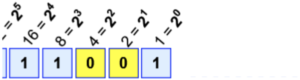
- Example: what is 99 in binary notation?

```
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





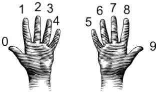


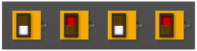
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

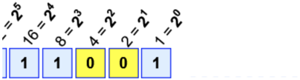
- Example: what is 99 in binary notation?

```
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





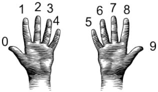


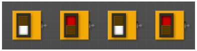
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

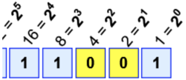
- Example: what is 99 in binary notation?

```
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





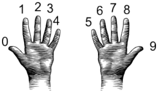


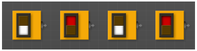
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

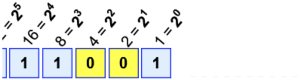
- Example: what is 99 in binary notation?

```
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





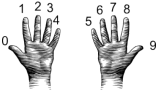


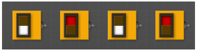
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

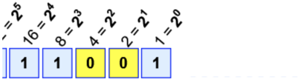
- Example: what is 99 in binary notation?

```
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





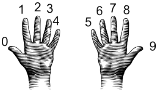


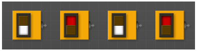
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

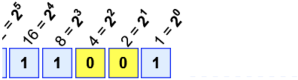
- Example: what is 99 in binary notation?

```
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







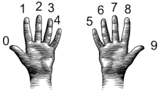
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

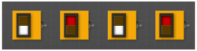
- Example: what is 99 in binary notation?

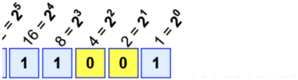
```
99/128 is 0 rem 99. First digit is 0:     0...
99/64 is 1 rem 35. Next digit is 1:       01...
35/32 is 1 rem 3. Next digit is 1:        011...
3/16 is 0 rem 3. Next digit is 0:         0110...
3/8 is 0 rem 3. Next digit is 0:          01100...
3/4 is 0 remainder 3. Next digit is 0:    011000...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



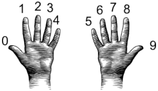


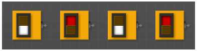


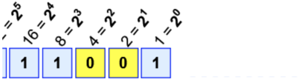
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

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

Binary to Decimal: Converting Between Bases



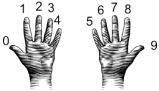


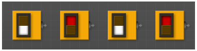


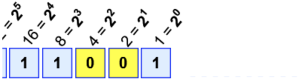
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

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

Binary to Decimal: Converting Between Bases



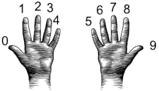


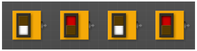


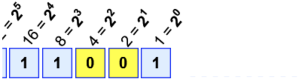
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

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

Binary to Decimal: Converting Between Bases



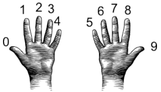


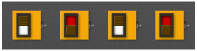


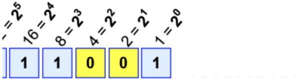
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

- 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



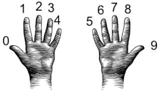


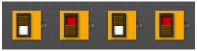


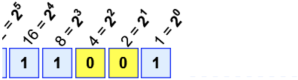
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

- 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



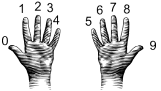


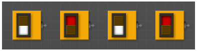


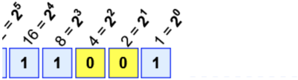
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

- 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



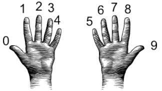


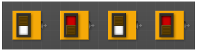


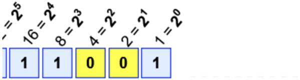
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

- 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



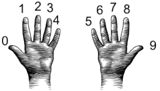


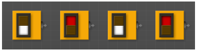


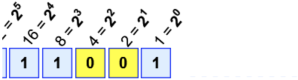
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

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

Binary to Decimal: Converting Between Bases



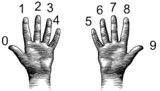


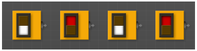


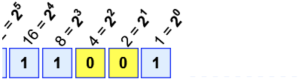
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

- 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.

Binary to Decimal: Converting Between Bases





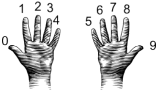


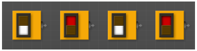
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

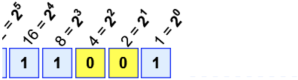
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by $2 = 2^1$. Add to sum.
  - Multiply next digit by $4 = 2^2$. Add to sum.
  - Multiply next digit by $8 = 2^3$. Add to sum.
  - Multiply next digit by $16 = 2^4$. Add to sum.
  - Multiply next digit by $32 = 2^5$. Add to sum.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





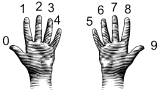


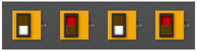
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

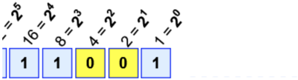
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by $2 = 2^1$. Add to sum.
  - Multiply next digit by $4 = 2^2$. Add to sum.
  - Multiply next digit by $8 = 2^3$. Add to sum.
  - Multiply next digit by $16 = 2^4$. Add to sum.
  - Multiply next digit by $32 = 2^5$. Add to sum.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

Binary to Decimal: Converting Between Bases





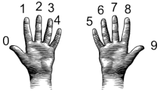


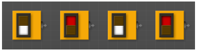
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

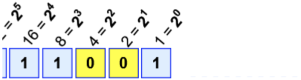
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by $2 = 2^1$. Add to sum.
  - Multiply next digit by $4 = 2^2$. Add to sum.
  - Multiply next digit by $8 = 2^3$. Add to sum.
  - Multiply next digit by $16 = 2^4$. Add to sum.
  - Multiply next digit by $32 = 2^5$. Add to sum.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

Binary to Decimal: Converting Between Bases





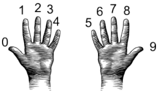


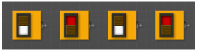
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

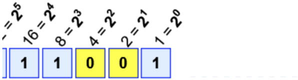
- From binary to decimal:
  - Set sum $=$ last digit.
  - Multiply next digit by $2 = 2^1$. Add to sum.
  - Multiply next digit by $4 = 2^2$. Add to sum.
  - Multiply next digit by $8 = 2^3$. Add to sum.
  - Multiply next digit by $16 = 2^4$. Add to sum.
  - Multiply next digit by $32 = 2^5$. Add to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

Binary to Decimal: Converting Between Bases





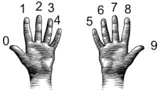


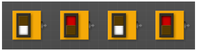
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

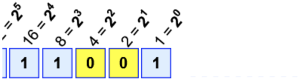
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by $2 = 2^1$. Add to sum.
  - Multiply next digit by $4 = 2^2$. Add to sum.
  - Multiply next digit by $8 = 2^3$. Add to sum.
  - Multiply next digit by $16 = 2^4$. Add to sum.
  - Multiply next digit by $32 = 2^5$. Add to sum.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

Binary to Decimal: Converting Between Bases





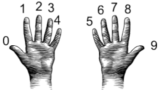


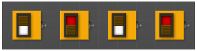
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

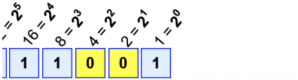
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by $2 = 2^1$. Add to sum.
  - Multiply next digit by $4 = 2^2$. Add to sum.
  - Multiply next digit by $8 = 2^3$. Add to sum.
  - Multiply next digit by $16 = 2^4$. Add to sum.
  - Multiply next digit by $32 = 2^5$. Add to sum.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

Binary to Decimal: Converting Between Bases





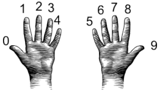


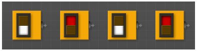
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

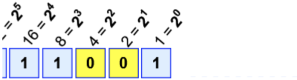
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by $2 = 2^1$. Add to sum.
  - Multiply next digit by $4 = 2^2$. Add to sum.
  - Multiply next digit by $8 = 2^3$. Add to sum.
  - Multiply next digit by $16 = 2^4$. Add to sum.
  - Multiply next digit by $32 = 2^5$. Add to sum.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

Binary to Decimal: Converting Between Bases





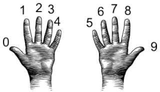


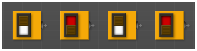
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

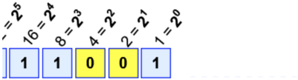
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by $2 = 2^1$. Add to sum.
  - Multiply next digit by $4 = 2^2$. Add to sum.
  - Multiply next digit by $8 = 2^3$. Add to sum.
  - Multiply next digit by $16 = 2^4$. Add to sum.
  - Multiply next digit by $32 = 2^5$. Add to sum.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

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

Binary to Decimal: Converting Between Bases





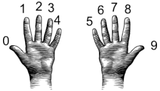


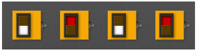
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

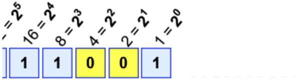
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by $2 = 2^1$. Add to sum.
  - Multiply next digit by $4 = 2^2$. Add to sum.
  - Multiply next digit by $8 = 2^3$. Add to sum.
  - Multiply next digit by $16 = 2^4$. Add to sum.
  - Multiply next digit by $32 = 2^5$. Add to sum.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

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

Binary to Decimal: Converting Between Bases





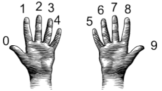


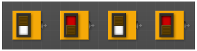
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

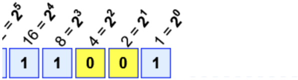
- From binary to decimal:
  - Set sum $=$ last digit.
  - Multiply next digit by $2 = 2^1$. Add to sum.
  - Multiply next digit by $4 = 2^2$. Add to sum.
  - Multiply next digit by $8 = 2^3$. Add to sum.
  - Multiply next digit by $16 = 2^4$. Add to sum.
  - Multiply next digit by $32 = 2^5$. Add to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





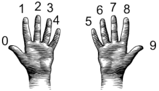


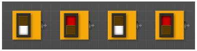
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

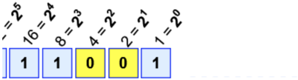
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by $2 = 2^1$. Add to sum.
  - Multiply next digit by $4 = 2^2$. Add to sum.
  - Multiply next digit by $8 = 2^3$. Add to sum.
  - Multiply next digit by $16 = 2^4$. Add to sum.
  - Multiply next digit by $32 = 2^5$. Add to sum.
  - 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:  61
```

Binary to Decimal: Converting Between Bases





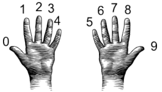


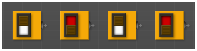
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

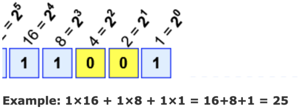
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by $2 = 2^1$. Add to sum.
  - Multiply next digit by $4 = 2^2$. Add to sum.
  - Multiply next digit by $8 = 2^3$. Add to sum.
  - Multiply next digit by $16 = 2^4$. Add to sum.
  - Multiply next digit by $32 = 2^5$. Add to sum.
  - 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:  61
```

Binary to Decimal: Converting Between Bases





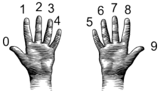


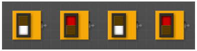
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

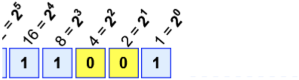
- Example: What is 10100100 in decimal?

  `Sum starts with:`

Binary to Decimal: Converting Between Bases





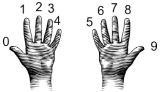


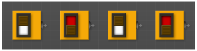
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

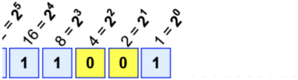
- Example: What is 10100100 in decimal?

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

Binary to Decimal: Converting Between Bases





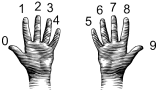


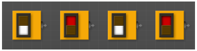
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

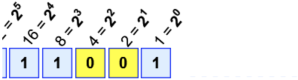
- Example: What is 10100100 in decimal?

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

Binary to Decimal: Converting Between Bases





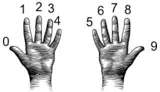


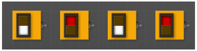
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

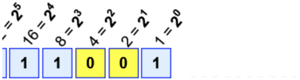
- Example: What is 10100100 in decimal?

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

Binary to Decimal: Converting Between Bases





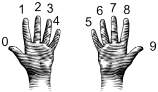


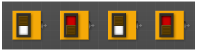
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

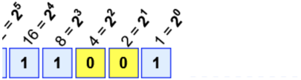
- Example: What is 10100100 in decimal?

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

Binary to Decimal: Converting Between Bases





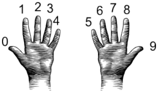


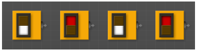
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

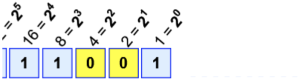
- Example: What is 10100100 in decimal?

```
Sum starts with:            0
0*2 = 0.  Add 0 to sum:     0
1*4 = 4.  Add 4 to sum:     4
0*8 = 0.  Add 0 to sum:
```

Binary to Decimal: Converting Between Bases





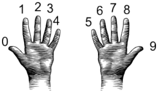


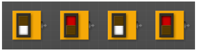
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

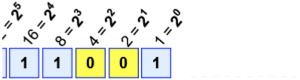
- Example: What is 10100100 in decimal?

```
Sum starts with:                0
0*2 = 0.   Add 0 to sum:        0
1*4 = 4.   Add 4 to sum:        4
0*8 = 0.   Add 0 to sum:        4
```

Binary to Decimal: Converting Between Bases





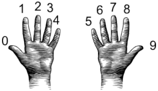


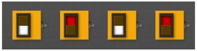
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

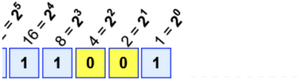
- Example: What is 10100100 in decimal?

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

Binary to Decimal: Converting Between Bases





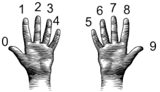


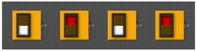
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

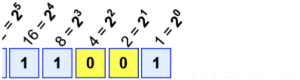
- Example: What is 10100100 in decimal?

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

Binary to Decimal: Converting Between Bases





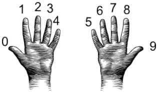


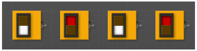
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

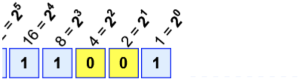
- Example: What is 10100100 in decimal?

```
Sum starts with:              0
0*2 = 0.   Add 0 to sum:      0
1*4 = 4.   Add 4 to sum:      4
0*8 = 0.   Add 0 to sum:      4
0*16 = 0.   Add 0 to sum:     4
1*32 = 32.   Add 32 to sum:
```

Binary to Decimal: Converting Between Bases





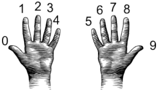


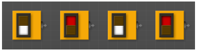
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

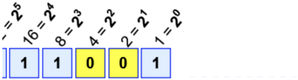
- Example: What is 10100100 in decimal?

```
Sum starts with:            0
0*2 = 0.   Add 0 to sum:    0
1*4 = 4.   Add 4 to sum:    4
0*8 = 0.   Add 0 to sum:    4
0*16 = 0.   Add 0 to sum:   4
1*32 = 32.   Add 32 to sum: 36
```

Binary to Decimal: Converting Between Bases





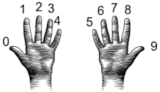


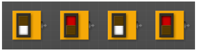
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

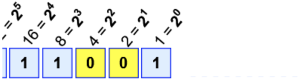
- Example: What is 10100100 in decimal?

```
Sum starts with:             0
0*2 = 0.   Add 0 to sum:     0
1*4 = 4.   Add 4 to sum:     4
0*8 = 0.   Add 0 to sum:     4
0*16 = 0.   Add 0 to sum:    4
1*32 = 32.   Add 32 to sum:  36
0*64 = 0.   Add 0 to sum:
```

Binary to Decimal: Converting Between Bases





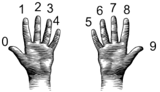


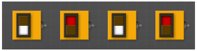
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

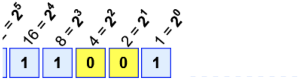
- Example: What is 10100100 in decimal?

```
Sum starts with:              0
0*2 = 0.   Add 0 to sum:      0
1*4 = 4.   Add 4 to sum:      4
0*8 = 0.   Add 0 to sum:      4
0*16 = 0.   Add 0 to sum:     4
1*32 = 32.   Add 32 to sum:   36
0*64 = 0.   Add 0 to sum:     36
```

Binary to Decimal: Converting Between Bases





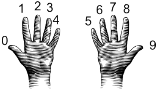


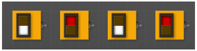
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

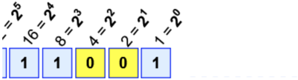
- Example: What is 10100100 in decimal?

```
Sum starts with:               0
0*2 = 0.  Add 0 to sum:        0
1*4 = 4.  Add 4 to sum:        4
0*8 = 0.  Add 0 to sum:        4
0*16 = 0.  Add 0 to sum:       4
1*32 = 32.  Add 32 to sum:     36
0*64 = 0.  Add 0 to sum:       36
1*128 = 0.  Add 128 to sum:
```

Binary to Decimal: Converting Between Bases





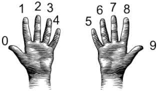


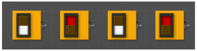
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

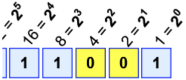
- Example: What is 10100100 in decimal?

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

Binary to Decimal: Converting Between Bases







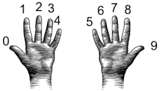
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

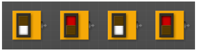
- Example: What is 10100100 in decimal?

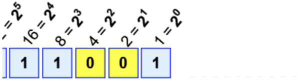
```
Sum starts with:                0
0*2 = 0.   Add 0 to sum:        0
1*4 = 4.   Add 4 to sum:        4
0*8 = 0.   Add 0 to sum:        4
0*16 = 0.   Add 0 to sum:       4
1*32 = 32.   Add 32 to sum:     36
0*64 = 0.   Add 0 to sum:       36
1*128 = 0.   Add 128 to sum:    164
```

The answer is 164.

Design Challenge: Incrementers



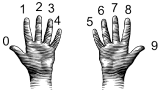


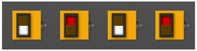


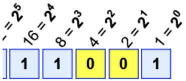
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

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

Design Challenge: Incrementers



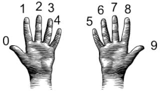


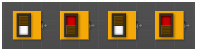


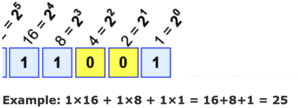
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

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

Design Challenge: Incrementers



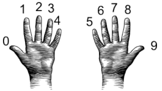


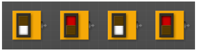


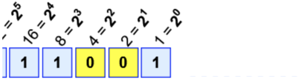
- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

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

Design Challenge: Incrementers







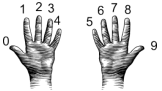
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

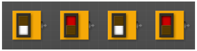
- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

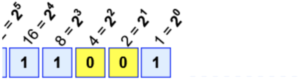
```
def addOne(n):
    m = n+1
    return(m)
```

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

Design Challenge: Incrementers







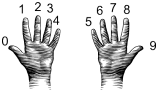
Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

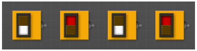
- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

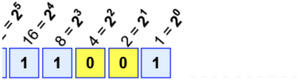
```
def addOne(n):
    m = n+1
    return(m)
```

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

Design Challenge: Incrementers







Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

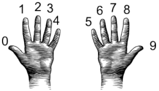
- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

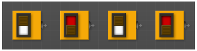
```
def addOne(n):
    m = n+1
    return(m)
```

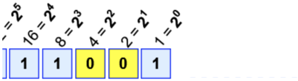
- Challenge: Write an algorithm for incrementing numbers expressed as words.
  Example: `"forty one"` → `"forty two"`

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

Design Challenge: Incrementers







Example: 1×16 + 1×8 + 1×1 = 16+8+1 = 25

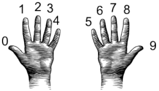
- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

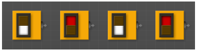
```
def addOne(n):
    m = n+1
    return(m)
```

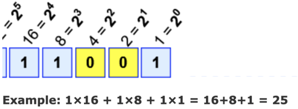
- Challenge: Write an algorithm for incrementing numbers expressed as words.
  Example: "forty one" → "forty two"

  Hint: Convert to numbers, increment, and convert back to strings.
- Challenge: Write an algorithm for incrementing binary numbers.

Design Challenge: Incrementers







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

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

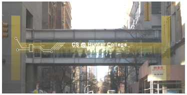
- Challenge: Write an algorithm for incrementing numbers expressed as words.
  Example: `"forty one"` → `"forty two"`

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

- Challenge: Write an algorithm for incrementing binary numbers.

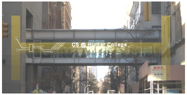
  Example: `"1001"` → `"1010"`

Recap



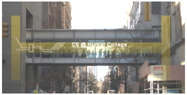
- Searching through data is a common task– built-in functions and standard design patterns for this.

Recap



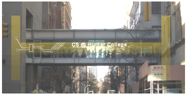
- Searching through data is a common task– built-in functions and standard design patterns for this.
- Programming languages can be classified by the level of abstraction and direct access to data.

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

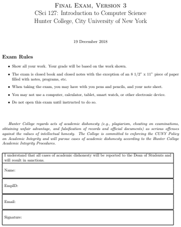
- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
  - With notes, examples, programs: what will help you on the exam.
  - No origami– it's distracting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

19 December 2018

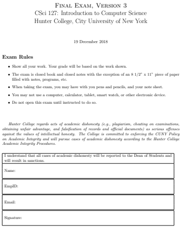
Exam Rules

- 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 1/2" x 11" piece of paper filled with notes, programs, etc.
- 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 on examinations, obtaining unfair advantage, and falsification of records and official documents) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

I understand that all cases of academic dishonesty will be reported to the Dean of Students and will result in sanctions.
Name:
EmpID:
Email:
Signature:

Exam Options



Final Exam, Version 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

19 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 1/2" x 11" piece of paper filled with notes, programs, etc.
- 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 on examinations, obtaining unfair advantage, and falsification of records and official documents) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

I understand that all cases of academic dishonesty will be reported to the Dean of Students and will result in sanctions.
Name:
EmpID:
Email:
Signature:

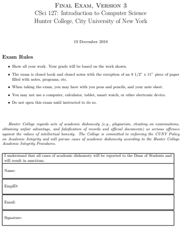
Exam Times:

- Default Regular Time: Monday, December 19, 9-11am.

Exam Options

Exam Times:

- Default Regular Time: Monday, December 19, 9-11am.
- Alternate Time: Friday, 16 December, 8am-10am.



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

19 December 2018

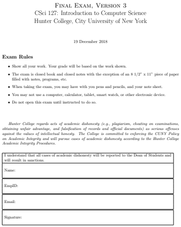
Exam Rules

- 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 1/2" x 11" piece of paper filled with notes, programs, etc.
- 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 on examinations, obtaining unfair advantage, and falsification of records and official documents) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

I understand that all cases of academic dishonesty will be reported to the Dean of Students and will result in sanctions.
Name:
EmpID:
Email:
Signature:

Exam Options



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

19 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 1/2" x 11" piece of paper filled with notes, programs, etc.
- 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 on examinations, obtaining unfair advantage, and falsification of records and official documents) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

I understand that all cases of academic dishonesty will be reported to the Dean of Students and will result in sanctions.

Name:

EmpID:

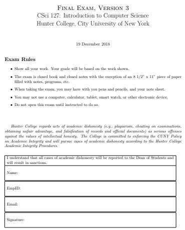
Email:

Signature:

Exam Times:

- Default Regular Time: Monday, December 19, 9-11am.
- Alternate Time: Friday, 16 December, 8am-10am.
- Accessibility Testing: Paperwork required. Must be completed on 5 December. If you have not done so already, email me no later than 5 December.

Exam Options



Final Exam, Version 3
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

19 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 1/2" x 11" piece of paper filled with notes, programs, etc.
- 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 on examinations, obtaining unfair advantage, and falsification of records and official documents) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

I understand that all cases of academic dishonesty will be reported to the Dean of Students and will result in sanctions.

Name:

EmpID:

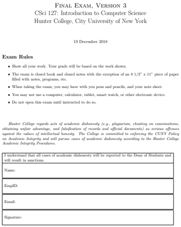
Email:

Signature:

Exam Times:

- Default Regular Time: Monday, December 19, 9-11am.
- Alternate Time: Friday, 16 December, 8am-10am.
- Accessibility Testing: Paperwork required. Must be completed on 5 December. If you have not done so already, email me no later than 5 December.
- Survey for your exam date choice will be available next lecture. No survey answer implies you will take the exam on December 19.

Exam Options



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

19 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 1/2" x 11" piece of paper filled with notes, programs, etc.
- 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 on examinations, obtaining unfair advantage, and falsification of records and official documents) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

I understand that all cases of academic dishonesty will be reported to the Dean of Students and will result in sanctions.

Name:

EmpID:

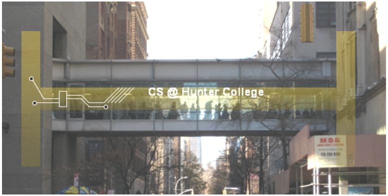
Email:

Signature:

Exam Times:

- Default Regular Time: Monday, December 19, 9-11am.
- Alternate Time: Friday, 16 December, 8am-10am.
- Accessibility Testing: Paperwork required. Must be completed on 5 December. If you have not done so already, email me no later than 5 December.
- Survey for your exam date choice will be available next lecture. No survey answer implies you will take the exam on December 19.
- If you choose to take the early date, you will not be given access to the exam on December 19 even if you miss the early exam.

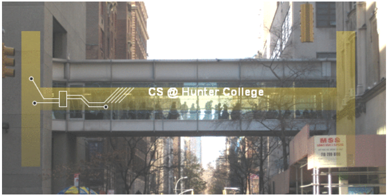
Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab

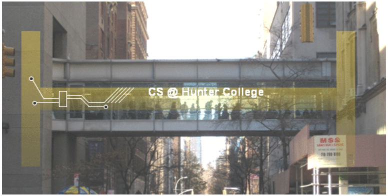
Weekly Reminders!



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

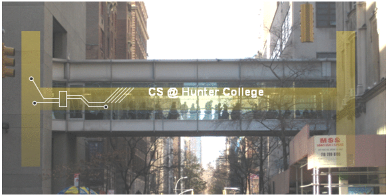
Weekly Reminders!



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

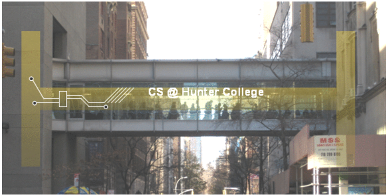
Weekly Reminders!



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)**

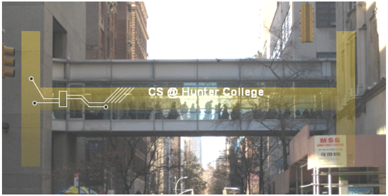
Weekly Reminders!



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

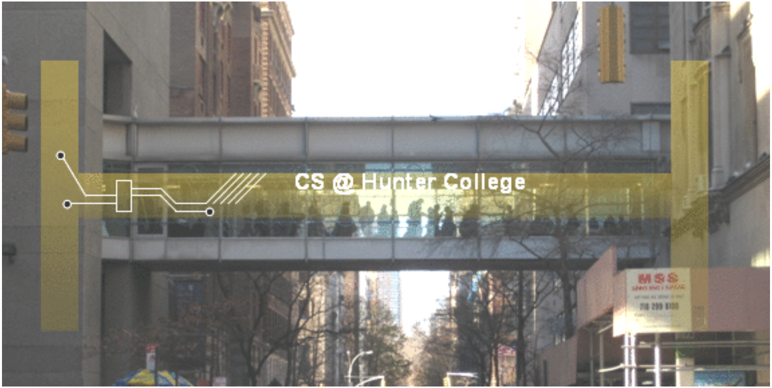
Weekly Reminders!



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.