

# CSci 127: Introduction to Computer Science



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

# Frequently Asked Questions

From lecture slips & recitation sections.

- When is the final?

# Frequently Asked Questions

From lecture slips & recitation sections.

- When is the final?

*Wednesday, 19 December, 9am-11am.*

# Frequently Asked Questions

From lecture slips & recitation sections.

- When is the final?  
*Wednesday, 19 December, 9am-11am.*
- I have another final then. What do I do?

# Frequently Asked Questions

From lecture slips & recitation sections.

- When is the final?

*Wednesday, 19 December, 9am-11am.*

- I have another final then. What do I do?

*We are arranging an alternative time (most likely reading day).*

# Frequently Asked Questions

From lecture slips & recitation sections.

- When is the final?

*Wednesday, 19 December, 9am-11am.*

- I have another final then. What do I do?

*We are arranging an alternative time (most likely reading day).*

- Do I have to take the final?

# Frequently Asked Questions

From lecture slips & recitation sections.

- When is the final?

*Wednesday, 19 December, 9am-11am.*

- I have another final then. What do I do?

*We are arranging an alternative time (most likely reading day).*

- Do I have to take the final?

*Yes, you have to pass the final (60 out of 100 points) to the pass the class.*

# Frequently Asked Questions

From lecture slips & recitation sections.

- When is the final?

*Wednesday, 19 December, 9am-11am.*

- I have another final then. What do I do?

*We are arranging an alternative time (most likely reading day).*

- Do I have to take the final?

*Yes, you have to pass the final (60 out of 100 points) to the pass the class.*

- Can I take the course No Credit/Credit?

# Frequently Asked Questions

From lecture slips & recitation sections.

- When is the final?

*Wednesday, 19 December, 9am-11am.*

- I have another final then. What do I do?

*We are arranging an alternative time (most likely reading day).*

- Do I have to take the final?

*Yes, you have to pass the final (60 out of 100 points) to the pass the class.*

- Can I take the course No Credit/Credit?

*Yes. We'll have forms ready after Thanksgiving Break.*

# Frequently Asked Questions

From lecture slips & recitation sections.

- When is the final?

*Wednesday, 19 December, 9am-11am.*

- I have another final then. What do I do?

*We are arranging an alternative time (most likely reading day).*

- Do I have to take the final?

*Yes, you have to pass the final (60 out of 100 points) to the pass the class.*

- Can I take the course No Credit/Credit?

*Yes. We'll have forms ready after Thanksgiving Break.*

- I'd like to take more computer science. What's next?

# Frequently Asked Questions

From lecture slips & recitation sections.

- When is the final?

*Wednesday, 19 December, 9am-11am.*

- I have another final then. What do I do?

*We are arranging an alternative time (most likely reading day).*

- Do I have to take the final?

*Yes, you have to pass the final (60 out of 100 points) to the pass the class.*

- Can I take the course No Credit/Credit?

*Yes. We'll have forms ready after Thanksgiving Break.*

- I'd like to take more computer science. What's next?

*Fabulous! The next courses are:*

- ▶ *CSci 135/136: Programming in C++.*

*Lecture: M, W, Th, 12:10-1pm; Sections: see schedule.*

- ▶ *CSci 150: Discrete structures (math for computing).*

*Lecture: M, Th, 1:10-2:25pm; Sections: see schedule.*

# Today's Topics



- Recap: folium and indefinite loops
- Design Patterns: Searching Data
- Data Representation
- Machine Language

# In Pairs or Triples:

*What does this code do?*

```
import folium
import pandas as pd

cuny = pd.read_csv('cunyLocations.csv')
mapCUNY = folium.Map(location=[40.75, -74.125])

for index, row in cuny.iterrows():
    lat = row["Latitude"]
    lon = row["Longitude"]
    name = row["Campus"]
    if row["College or Institution Type"] == "Senior Colleges":
        collegeIcon = folium.Icon(color="purple")
    else:
        collegeIcon = folium.Icon(color="blue")
    newMarker = folium.Marker([lat, lon], popup=name, icon=collegeIcon)
    newMarker.add_to(mapCUNY)

mapCUNY.save(outfile='cunyLocationsSenior.html')
```

# folium example

*What does this code do?*

```
import folium
import pandas as pd

cuny = pd.read_csv('cunyLocations.csv')
mapCUNY = folium.Map(location=[40.75, -74.125])

for index, row in cuny.iterrows():
    lat = row["Latitude"]
    lon = row["Longitude"]
    name = row["Campus"]
    if row["College or Institution Type"] == "Senior Colleges":
        collegeIcon = folium.Icon(color="purple")
    else:
        collegeIcon = folium.Icon(color="blue")
    newMarker = folium.Marker([lat, lon], popup=name, icon=collegeIcon)
    newMarker.add_to(mapCUNY)

mapCUNY.save(outfile='cunyLocationsSenior.html')
```

# folium example

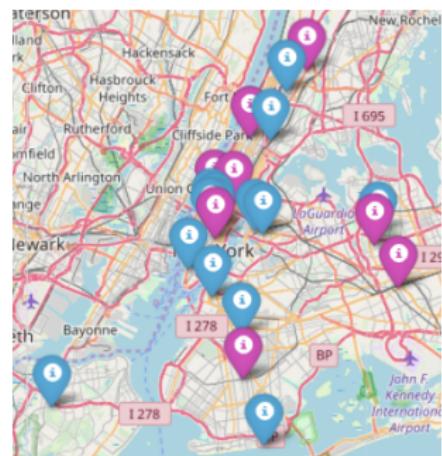
What does this code do?

```
import folium
import pandas as pd

cuny = pd.read_csv('cunyLocations.csv')
mapCUNY = folium.Map(location=[40.75, -74.125])

for index, row in cuny.iterrows():
    lat = row["Latitude"]
    lon = row["Longitude"]
    name = row["Campus"]
    if row["College or Institution Type"] == "Senior Colleges":
        collegeIcon = folium.Icon(color="purple")
    else:
        collegeIcon = folium.Icon(color="blue")
    newMarker = folium.Marker([lat, lon], popup=name, icon=collegeIcon)
    newMarker.add_to(mapCUNY)

mapCUNY.save(outfile='cunyLocationsSenior.html')
```



folium

- A module for making HTML maps.

## Folium



# folium

## Folium



- A module for making HTML maps.
- It's a Python interface to the popular `leaflet.js`.

# folium

Folium



- A module for making HTML maps.
- It's a Python interface to the popular `leaflet.js`.
- Outputs `.html` files which you can open in a browser.

# folium

Folium



- A module for making HTML maps.
- It's a Python interface to the popular `leaflet.js`.
- Outputs `.html` files which you can open in a browser.
- An extra step:

# folium

## Folium

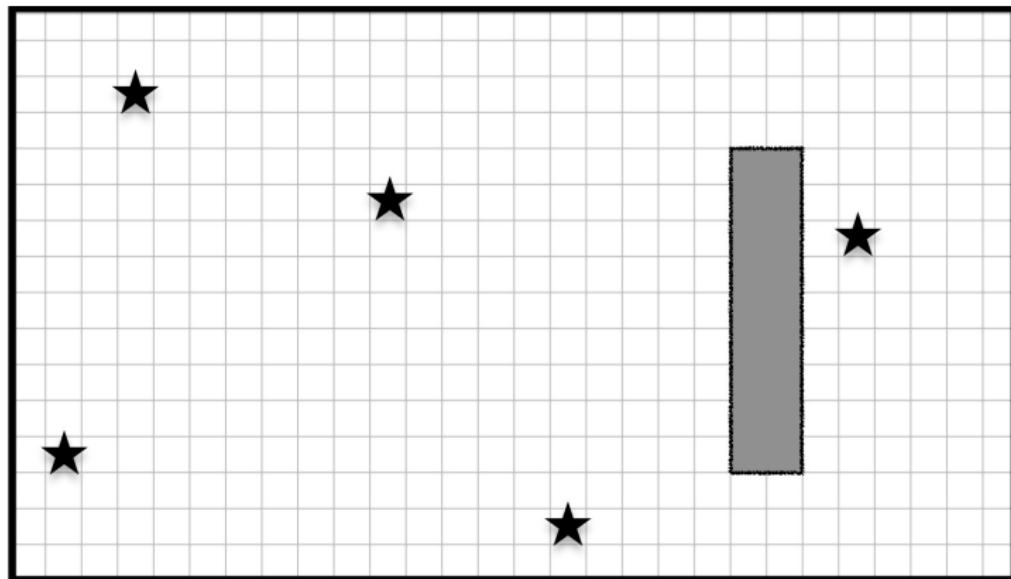


- A module for making HTML maps.
- It's a Python interface to the popular `leaflet.js`.
- Outputs `.html` files which you can open in a browser.
- An extra step:

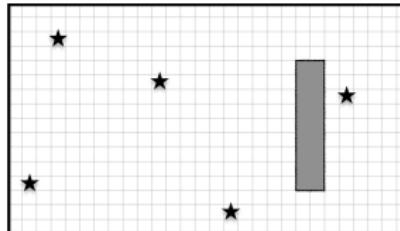
*Write code.* → *Run program.* → *Open .html in browser.*

# From Last Time: Design Challenge

Collect all five stars (locations randomly generated):

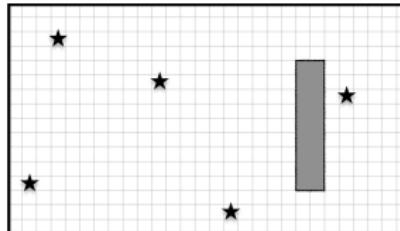


# From Last Time: Design Challenge



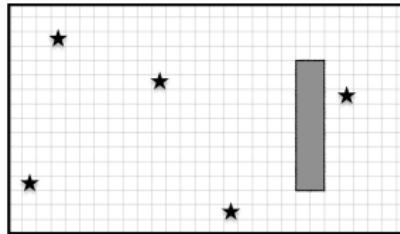
- Possible approaches:

# From Last Time: Design Challenge



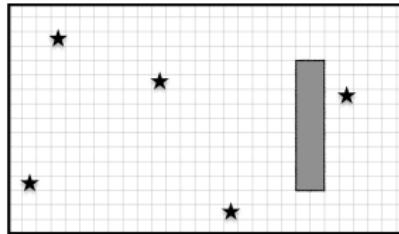
- Possible approaches:
  - ▶ Randomly wander until all 5 collected, or

# From Last Time: Design Challenge



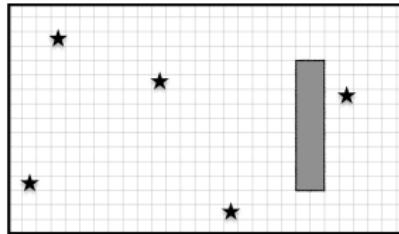
- Possible approaches:
  - ▶ Randomly wander until all 5 collected, or
  - ▶ Start in one corner, and systematically visit every point.

# From Last Time: Design Challenge



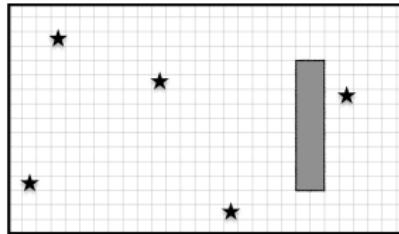
- Possible approaches:
  - ▶ Randomly wander until all 5 collected, or
  - ▶ Start in one corner, and systematically visit every point.
- **Input:** The map of the 'world.'

# From Last Time: Design Challenge



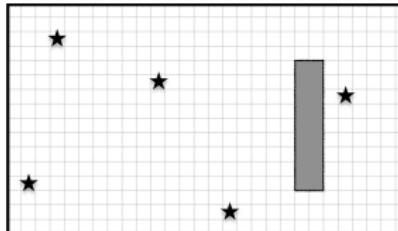
- Possible approaches:
  - ▶ Randomly wander until all 5 collected, or
  - ▶ Start in one corner, and systematically visit every point.
- **Input:** The map of the 'world.'
- **Output:** Time taken and/or locations of the 5 stars.

# From Last Time: Design Challenge



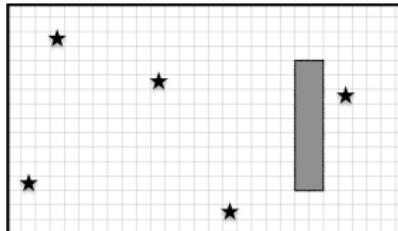
- Possible approaches:
  - ▶ Randomly wander until all 5 collected, or
  - ▶ Start in one corner, and systematically visit every point.
- **Input:** The map of the 'world.'
- **Output:** Time taken and/or locations of the 5 stars.
- How to store locations? Use `numpy array` with -1 everywhere.

# From Last Time: Design Challenge



- Possible approaches:
  - ▶ Randomly wander until all 5 collected, or
  - ▶ Start in one corner, and systematically visit every point.
- **Input:** The map of the 'world.'
- **Output:** Time taken and/or locations of the 5 stars.
- How to store locations? Use numpy array with -1 everywhere.
- Possible algorithms: while numStars < 5:

# From Last Time: Design Challenge



- Possible approaches:
  - ▶ Randomly wander until all 5 collected, or
  - ▶ Start in one corner, and systematically visit every point.
- **Input:** The map of the 'world.'
- **Output:** Time taken and/or locations of the 5 stars.
- How to store locations? Use `numpy` array with -1 everywhere.
- Possible algorithms: while `numStars < 5`:
  - ▶ Move forward.
  - ▶ If wall, mark 0 in map, randomly turn left or right.
  - ▶ If star, mark 1 in map and add 1 to `numStars`.
  - ▶ Otherwise, mark 2 in map that it's an empty square.
- If only turned left when you ran into a wall, what would happen?

# In Pairs or Triples:

*Predict what the code will do:*

```
#Random search
import turtle
import random
tess = turtle.Turtle()
tess.color('steelBlue')
tess.shape('turtle')
tess.penup()
#Start off screen:
tess.goto(-250,-250)
#Remember: abs(x) < 25 means absolute value: -25 < x < 25
while abs(tess.xcor()) > 25 or abs(tess.ycor()) > 25:
    x = random.randrange(-200,200)
    y = random.randrange(-200,200)
    tess.goto(x,y)
    tess.stamp()
    print(tess.xcor(), tess.ycor())
print('Found the center!')
```

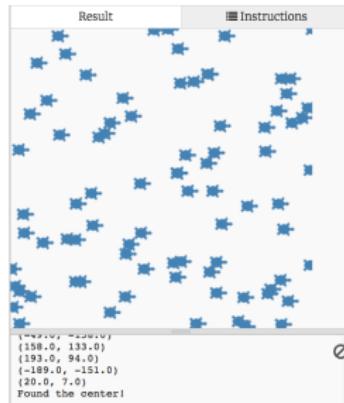
# Python Tutor

```
#Random search
import turtle
import random
tess = turtle.Turtle()
tess.color('steelBlue')
tess.shape('turtle')
tess.penup()
#Start off screen:
tess.goto(-250,-250)
#Then move until abs(x) < 25 means absolute value: -25 < x < 25
#and abs(y) > 25 or abs(y) < 25:
while abs(tess.xcor()) > 25 or abs(tess.ycor()) > 25:
    x = random.randrange(-200,200)
    y = random.randrange(-200,200)
    tess.goto(x,y)
    tess.stamp()
    print(tess.xcor(), tess.ycor())
print('Found the center!')
```

(Demo with trinket)

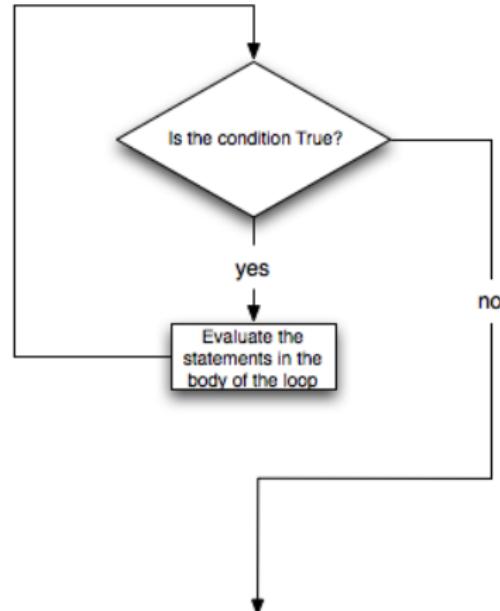
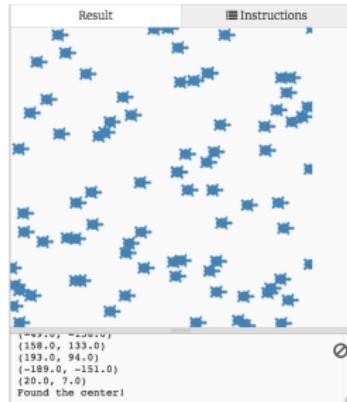
# Indefinite Loops

```
#Random search
import turtle
import random
tess = turtle.Turtle()
tess.color('steelBlue')
tess.shape('turtle')
tess.penup()
#Start off screen:
tess.goto(-250,-250)
#Remember: abs(x) < 25 means absolute value: -25 < x < 25
while abs(tess.xcor()) > 25 or abs(tess.ycor()) > 25:
    x = random.randrange(-200,200)
    y = random.randrange(-200,200)
    tess.goto(x,y)
    tess.stamp()
    print(tess.xcor(), tess.ycor())
print("Found the center!")
```



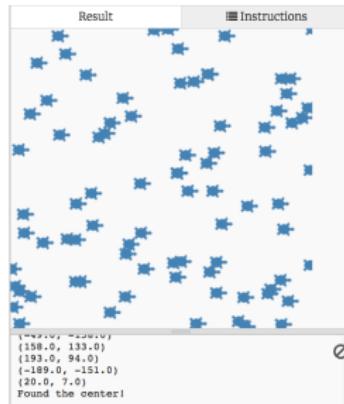
# Indefinite Loops

```
#Random search
import turtle
import random
tess = turtle.Turtle()
tess.color('steelBlue')
tess.shape('turtle')
tess.penup()
#Start off screen:
tess.goto(-250,-250)
#Remember: abs(x) < 25 means absolute value: -25 < x < 25
while abs(tess.xcor()) > 25 or abs(tess.ycor()) > 25:
    x = random.randrange(-200,200)
    y = random.randrange(-200,200)
    tess.goto(x,y)
    tess.stamp()
    print(tess.xcor(), tess.ycor())
print("Found the center!")
```



# Indefinite Loops

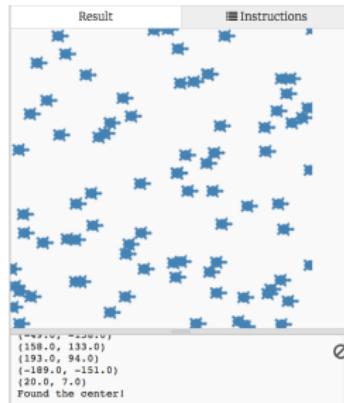
```
#Random search
import turtle
import random
tess = turtle.Turtle()
tess.color('steelBlue')
tess.shape('turtle')
tess.penup()
#Start off screen:
tess.goto(-250,-250)
#Remember: abs(x) < 25 means absolute value: -25 < x < 25
while abs(tess.xcor()) > 25 or abs(tess.ycor()) > 25:
    x = random.randrange(-200,200)
    y = random.randrange(-200,200)
    tess.goto(x,y)
    tess.stamp()
    print(tess.xcor(), tess.ycor())
print("Found the center!")
```



# Indefinite Loops

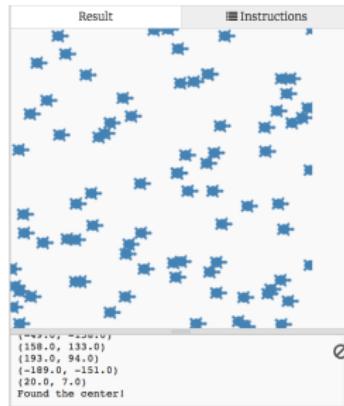
```
#Random search
import turtle
import random
tess = turtle.Turtle()
tess.color('steelBlue')
tess.shape('turtle')
tess.pensize(2)
#Start off screen:
tess.goto(-250,-250)
#Remember: abs(x) < 25 means absolute value: -25 < x < 25
while abs(tess.xcor()) > 25 or abs(tess.ycor()) > 25:
    x = random.randrange(-200,200)
    y = random.randrange(-200,200)
    tess.goto(x,y)
    tess.stamp()
    print(tess.xcor(), tess.ycor())
print("Found the center!")
```

- Indefinite loops repeat as long as the condition is true.



# Indefinite Loops

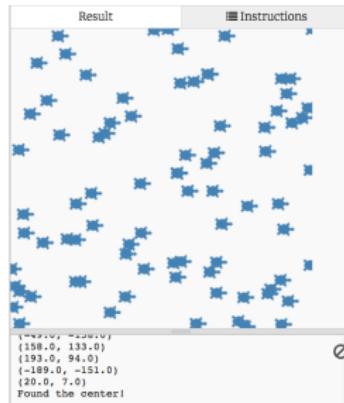
```
#Random search
import turtle
import random
tess = turtle.Turtle()
tess.color('steelBlue')
tess.shape('turtle')
tess.penup()
#Start off screen:
tess.goto(-250,-250)
#Remember: abs(x) < 25 means absolute value: -25 < x < 25
while abs(tess.xcor()) > 25 or abs(tess.ycor()) > 25:
    x = random.randrange(-200,200)
    y = random.randrange(-200,200)
    tess.goto(x,y)
    tess.stamp()
    print(tess.xcor(), tess.ycor())
print("Found the center!")
```



- Indefinite loops repeat as long as the condition is true.
- Could execute the body of the loop zero times, 10 times, infinite number of times.

# Indefinite Loops

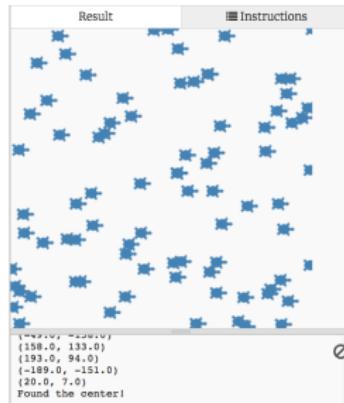
```
#Random search
import turtle
import random
tess = turtle.Turtle()
tess.color('steelBlue')
tess.shape('turtle')
tess.penup()
#Start off screen:
tess.goto(-250,-250)
#Remember: abs(x) < 25 means absolute value: -25 < x < 25
while abs(tess.xcor()) > 25 or abs(tess.ycor()) > 25:
    x = random.randrange(-200,200)
    y = random.randrange(-200,200)
    tess.goto(x,y)
    tess.stamp()
    print(tess.xcor(), tess.ycor())
print("Found the center!")
```



- Indefinite loops repeat as long as the condition is true.
- Could execute the body of the loop zero times, 10 times, infinite number of times.
- The condition determines how many times.

# Indefinite Loops

```
#Random search
import turtle
import random
tess = turtle.Turtle()
tess.color('steelBlue')
tess.shape('turtle')
tess.penup()
#Start off screen:
tess.goto(-250,-250)
#Remember: abs(x) < 25 means absolute value: -25 < x < 25
while abs(tess.xcor()) > 25 or abs(tess.ycor()) > 25:
    x = random.randrange(-200,200)
    y = random.randrange(-200,200)
    tess.goto(x,y)
    tess.stamp()
    print(tess.xcor(), tess.ycor())
print("Found the center!")
```



- Indefinite loops repeat as long as the condition is true.
- Could execute the body of the loop zero times, 10 times, infinite number of times.
- The condition determines how many times.
- Very useful for checking input, simulations, and games.

# Today's Topics



- Recap: folium and indefinite loops
- **Design Patterns: Searching Data**
- Data Representation
- Machine Language

# In Pairs or Triples:



*Answer the following questions on your lecture slip:*

Of the students in the room,

- Whose name comes first alphabetically?
- Whose name comes last alphabetically?
- Is there someone in the room with your initials?

## In Pairs or Triples:



Design a program that takes a CSV file and a set of initials:

- Whose name comes first alphabetically?
- Whose name comes last alphabetically?
- Is there someone in the room with your initials?

# Design Question: Find first alphabetically



- In Pandas, lovely built-in functions:

# Design Question: Find first alphabetically



- In Pandas, lovely built-in functions:
  - ▶ `df.sort_values('First Name')` and
  - ▶ `df['First Name'].min()`

# Design Question: Find first alphabetically



- In Pandas, lovely built-in functions:
  - ▶ `df.sort_values('First Name')` and
  - ▶ `df['First Name'].min()`
- What if you don't have a CSV and DataFrame, or data not ordered?

# Design Question: Find first alphabetically



- What if you don't have a CSV and DataFrame, or data not ordered?

# Design Question: Find first alphabetically



- What if you don't have a CSV and DataFrame, or data not ordered?
- Useful *Design Pattern*: min/max

# Design Question: Find first alphabetically



- What if you don't have a CSV and DataFrame, or data not ordered?
- Useful *Design Pattern*: min/max
  - ▶ Set a variable to worst value (i.e. `maxN = 0` or `first = "ZZ"`).

# Design Question: Find first alphabetically



- What if you don't have a CSV and DataFrame, or data not ordered?
- Useful *Design Pattern*: min/max
  - ▶ Set a variable to worst value (i.e. `maxN = 0` or `first = "ZZ"`).
  - ▶ For each item, X, in the list:

# Design Question: Find first alphabetically



- What if you don't have a CSV and DataFrame, or data not ordered?
- Useful *Design Pattern*: min/max
  - ▶ Set a variable to worst value (i.e. `maxN = 0` or `first = "ZZ"`).
  - ▶ For each item, X, in the list:
    - ★ Compare X to your variable.

# Design Question: Find first alphabetically



- What if you don't have a CSV and DataFrame, or data not ordered?
- Useful *Design Pattern*: min/max
  - ▶ Set a variable to worst value (i.e. `maxN = 0` or `first = "ZZ"`).
  - ▶ For each item, X, in the list:
    - ★ Compare X to your variable.
    - ★ If better, update your variable to be X.

# Design Question: Find first alphabetically



- What if you don't have a CSV and DataFrame, or data not ordered?
- Useful *Design Pattern*: min/max
  - ▶ Set a variable to worst value (i.e. `maxN = 0` or `first = "ZZ"`).
  - ▶ For each item, X, in the list:
    - ★ Compare X to your variable.
    - ★ If better, update your variable to be X.
  - ▶ Print/return X.

# Design Question: Find Matching Initials



- How do we stop, if we find a match?

# Design Question: Find Matching Initials



- How do we stop, if we find a match?
- Change the loop to be indefinite (i.e. while loop):
  - ▶ Set a variable to found = False

# Design Question: Find Matching Initials



- How do we stop, if we find a match?
- Change the loop to be indefinite (i.e. while loop):
  - ▶ Set a variable to `found = False`
  - ▶ while there are items in the list and not found

# Design Question: Find Matching Initials



- How do we stop, if we find a match?
- Change the loop to be indefinite (i.e. while loop):
  - ▶ Set a variable to `found = False`
  - ▶ while there are items in the list and not found
    - ★ If item matches your value, set `found = True`

# Design Question: Find Matching Initials



- How do we stop, if we find a match?
- Change the loop to be indefinite (i.e. while loop):
  - ▶ Set a variable to `found = False`
  - ▶ while there are items in the list and not found
    - ★ If item matches your value, set `found = True`
  - ▶ Print/return value.

## In Pairs or Triples:

*Predict what the code will do:*

```
nums = [1,4,10,6,5,42,9,8,12]
```

```
maxNum = 0
for n in nums:
    if n > maxNum:
        maxNum = n
print('The max is', maxNum)
```

# Python Tutor

```
nums = [1,4,10,6,5,42,9,8,12]  
  
maxNum = 0  
for n in nums:  
    if n > maxNum:  
        maxNum = n  
print('The max is', maxNum)
```

(Demo with pythonTutor)

# In Pairs or Triples:

Predict what the code will do:

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

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

# Python Tutor

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

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

(Demo with pythonTutor)

## In Pairs or Triples:

- Write a function that asks a user for number after 2000 but before 2018. The function should repeatedly ask the user for a number until they enter one within the range and return the number.

# Coding

- Write a function that asks a user for number after 2000 but before 2018. The function should repeatedly ask the user for a number until they enter one within the range and return the number..

# Coding

- Write a function that asks a user for number after 2000 but before 2018. The function should repeatedly ask the user for a number until they enter one within the range and return the number.

```
def getYear():
```

# Coding

- Write a function that asks a user for number after 2000 but before 2018. The function should repeatedly ask the user for a number until they enter one within the range and return the number.

```
def getYear():
```

```
    return(num)
```

# Coding

- Write a function that asks a user for number after 2000 but before 2018. The function should repeatedly ask the user for a number until they enter one within the range and return the number.

```
def getYear():
    num = 0

    return(num)
```

# Coding

- Write a function that asks a user for number after 2000 but before 2018. The function should repeatedly ask the user for a number until they enter one within the range and return the number.

```
def getYear():
    num = 0
    while num <= 2000 or num >= 2018:

        return(num)
```

# Coding

- Write a function that asks a user for number after 2000 but before 2018. The function should repeatedly ask the user for a number until they enter one within the range and return the number.

```
def getYear():
    num = 0
    while num <= 2000 or num >= 2018:
        num = int(input('Enter a number > 2000 & < 2018'))
    return(num)
```

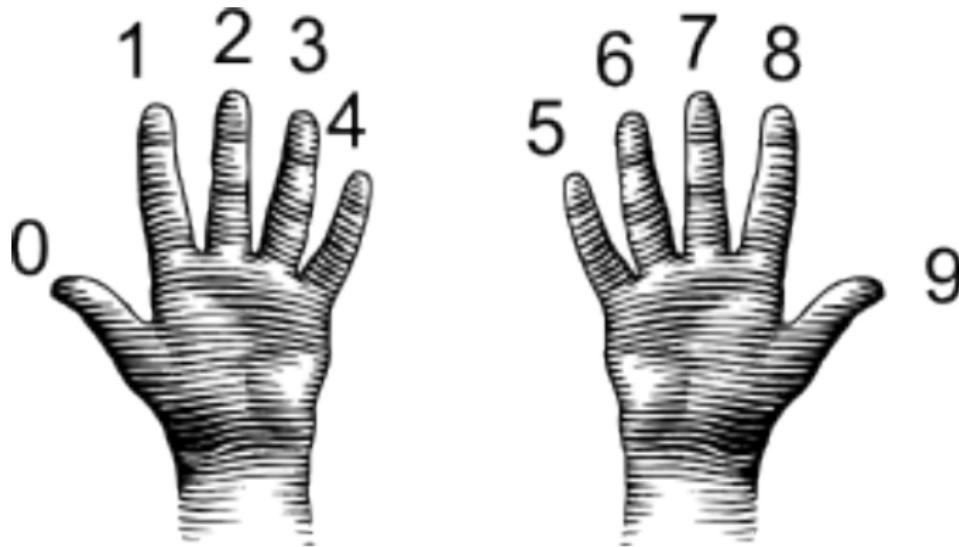
# Today's Topics



- Recap: folium and indefinite loops
- Design Patterns: Searching Data
- **Data Representation**
- Machine Language

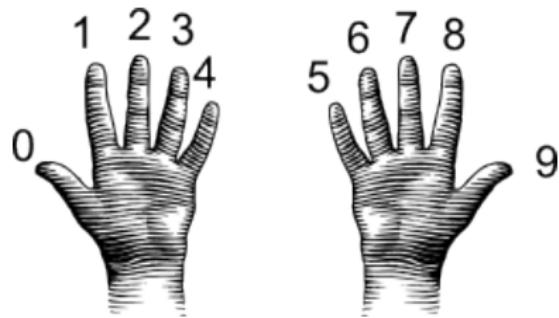
# Recall: Decimal & Hexadecimal Numbers

Counting with 10 digits:



(from i-programmer.info)

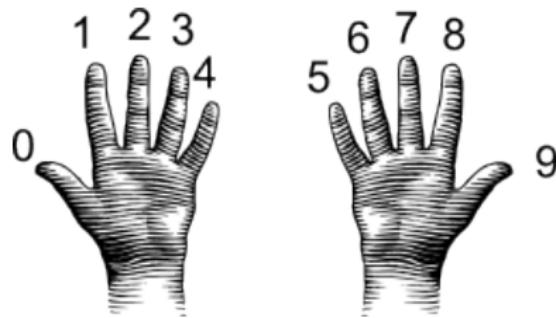
# Decimal



(from i-programmer.info)

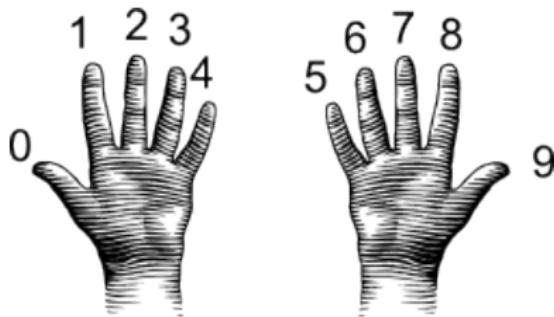
# Decimal

00 01 02 03 04 05 06 07 08 09



(from i-programmer.info)

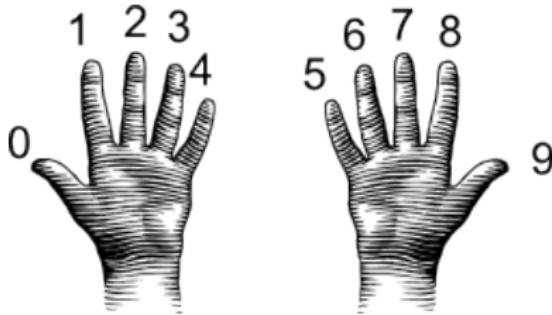
# Decimal



(from i-programmer.info)

00 01 02 03 04 05 06 07 08 09  
10 11 12 13 14 15 16 17 18 19

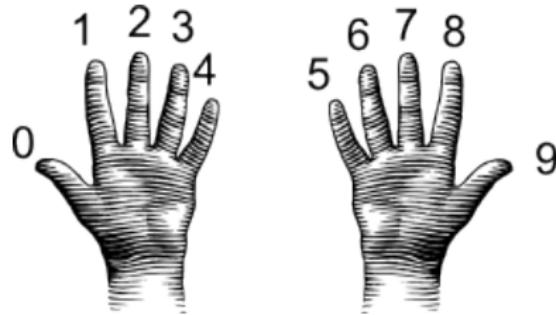
# Decimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29

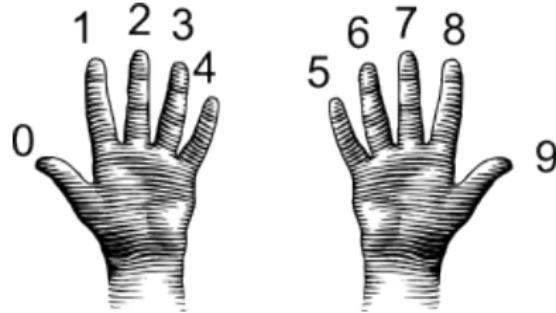
# Decimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

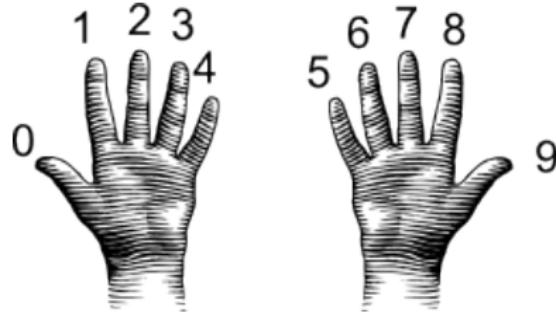
# Decimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49

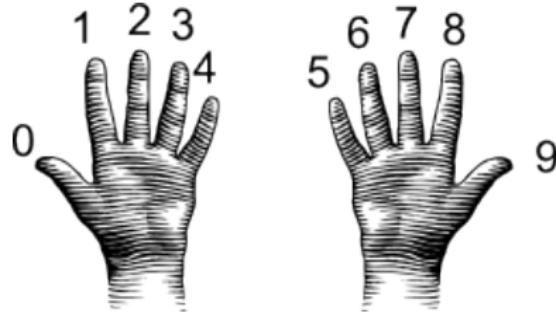
# Decimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59

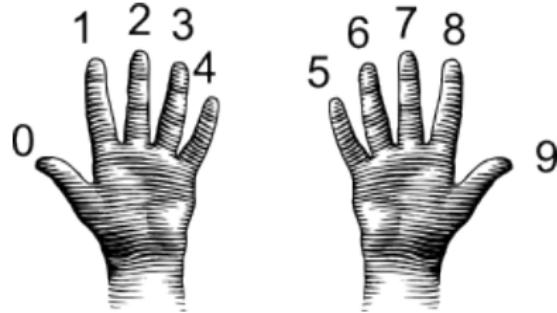
# Decimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69

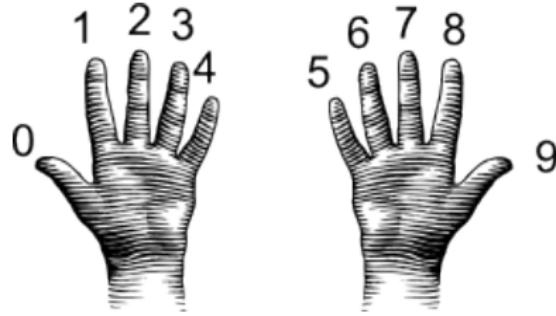
# Decimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79

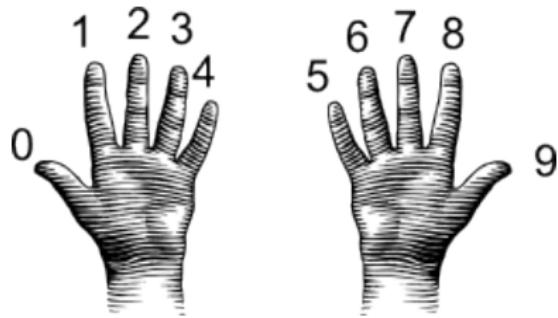
# Decimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89

# Decimal

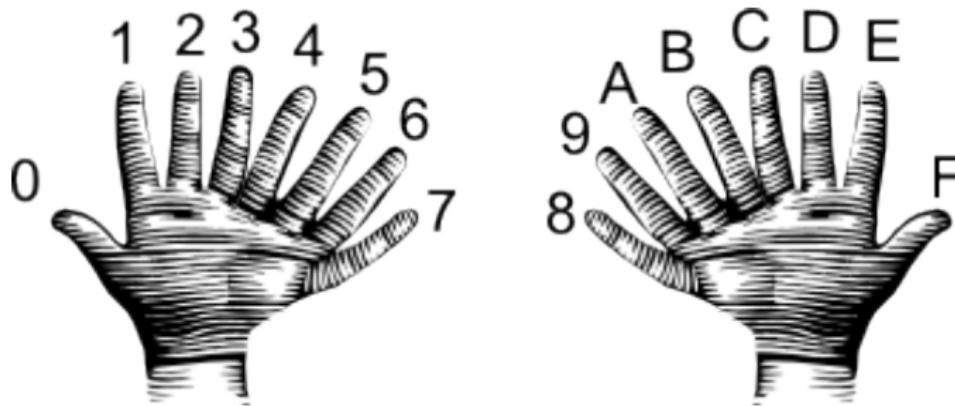


(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

# Recall: Decimal & Hexadecimal Numbers

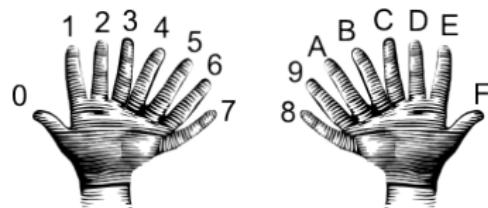
Counting with 16 digits:



(from i-programmer.info)

# Hexadecimal

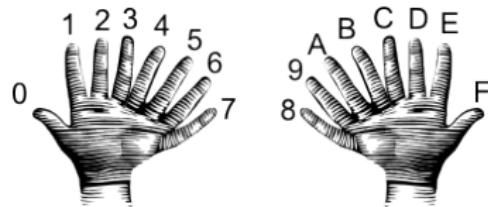
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F



(from i-programmer.info)

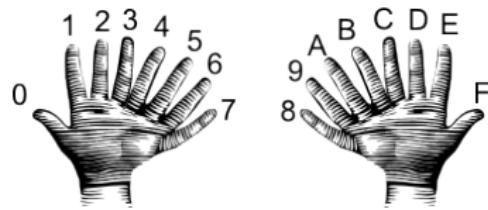
# Hexadecimal

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F



(from i-programmer.info)

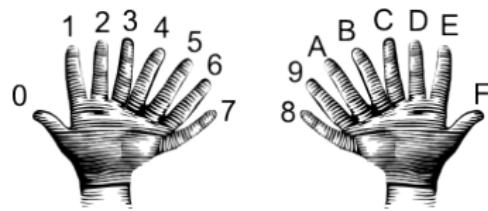
# Hexadecimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F

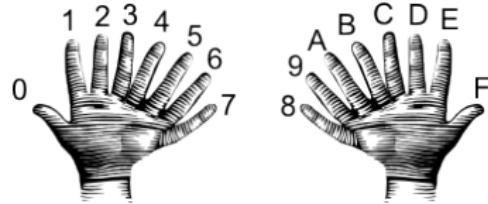
# Hexadecimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F

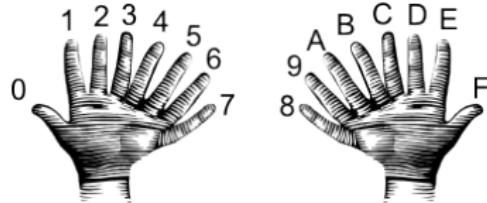
# Hexadecimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

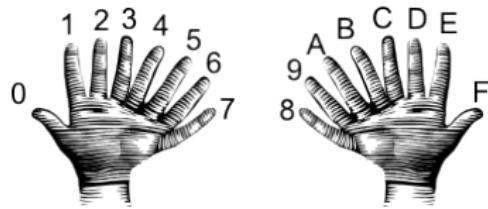
# Hexadecimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F

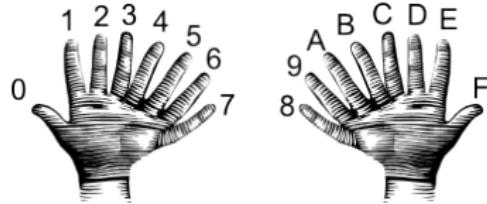
# Hexadecimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F

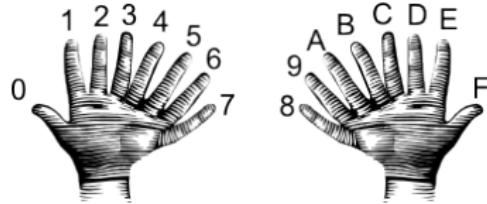
# Hexadecimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F

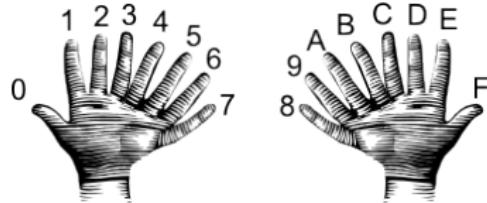
# Hexadecimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F

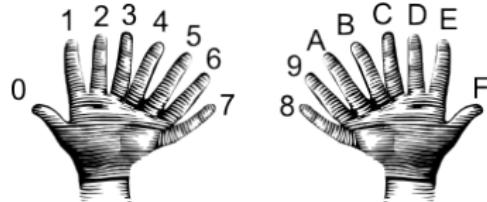
# Hexadecimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F

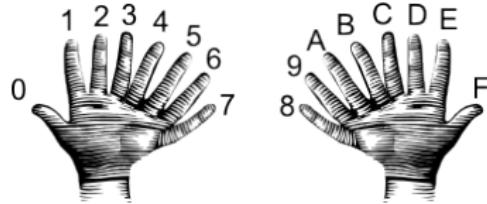
# Hexadecimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF

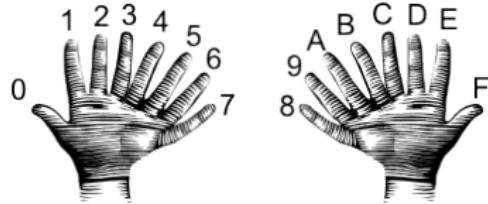
# Hexadecimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF

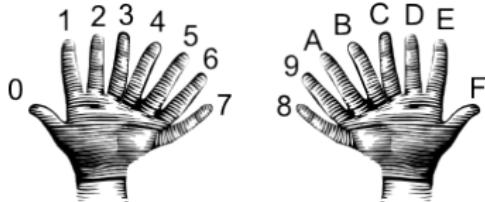
# Hexadecimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF

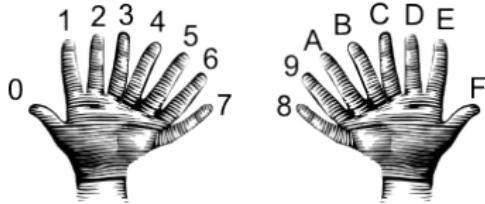
# Hexadecimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF

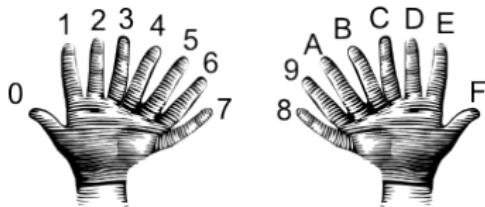
# Hexadecimal



(from i-programmer.info)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF

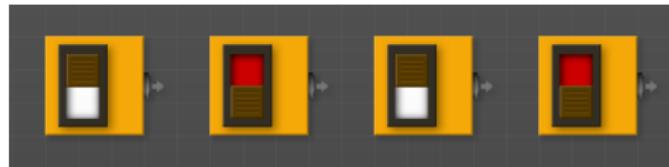
# Hexadecimal



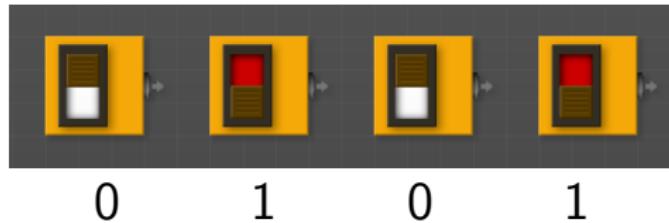
(from i-programmer.info)

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F  
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F  
40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F  
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F  
60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F  
70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F  
80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F  
90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F  
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF  
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF  
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF  
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF  
E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF  
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
```

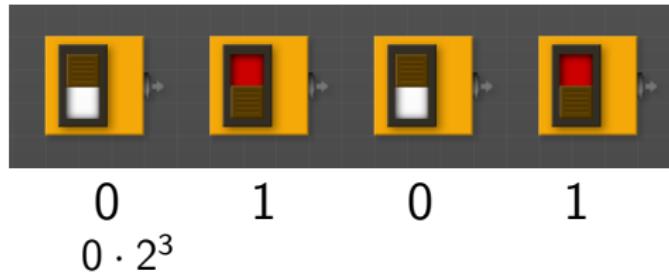
# Binary Numbers



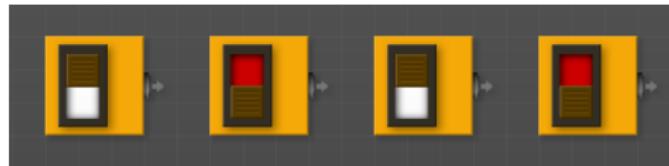
# Binary Numbers



# Binary Numbers



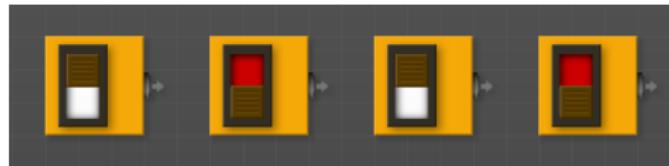
# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

- Only have two digits: 0 and 1.

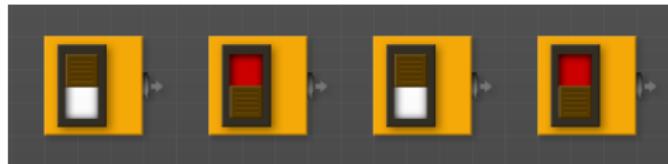
# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

- Only have two digits: 0 and 1.
- Can view as a series of switches that are either off (0) or on (1).

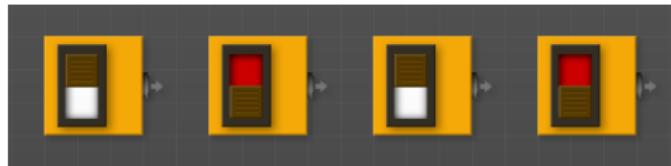
# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

- Only have two digits: 0 and 1.
- Can view as a series of switches that are either off (0) or on (1).
- 4-bit number uses 4 binary digits and ranges from 0000 or 0 to 1111 or  $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$

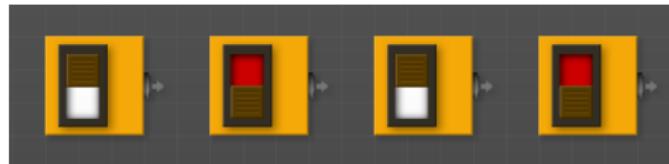
# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

- Only have two digits: 0 and 1.
- Can view as a series of switches that are either off (0) or on (1).
- 4-bit number uses 4 binary digits and ranges from 0000 or 0 to 1111 or  $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$
- Counting by 2's:

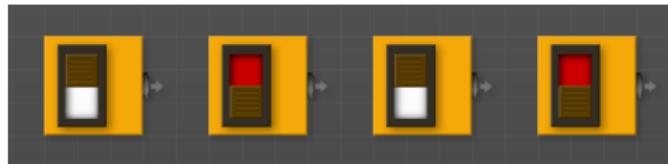
# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

- Only have two digits: 0 and 1.
- Can view as a series of switches that are either off (0) or on (1).
- 4-bit number uses 4 binary digits and ranges from 0000 or 0 to 1111 or  $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$
- Counting by 2's: 0

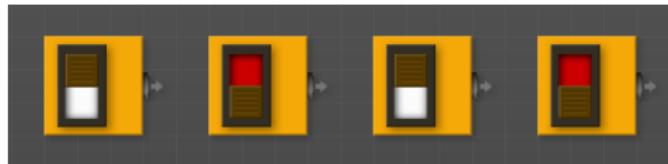
# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

- Only have two digits: 0 and 1.
- Can view as a series of switches that are either off (0) or on (1).
- 4-bit number uses 4 binary digits and ranges from 0000 or 0 to 1111 or  $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$
- Counting by 2's: 0 1

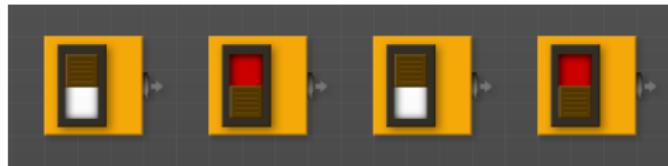
# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

- Only have two digits: 0 and 1.
- Can view as a series of switches that are either off (0) or on (1).
- 4-bit number uses 4 binary digits and ranges from 0000 or 0 to 1111 or  $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$
- Counting by 2's: 0 1 10

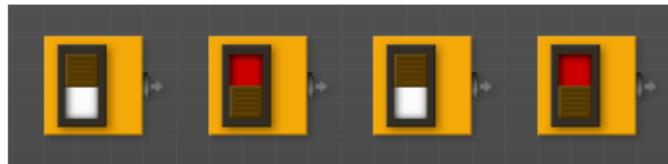
# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

- Only have two digits: 0 and 1.
- Can view as a series of switches that are either off (0) or on (1).
- 4-bit number uses 4 binary digits and ranges from 0000 or 0 to 1111 or  $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$
- Counting by 2's: 0 1 10 11

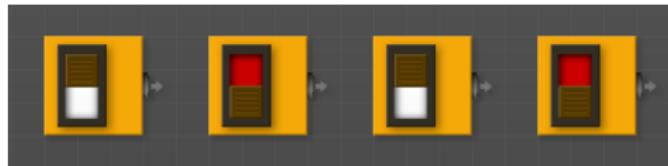
# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

- Only have two digits: 0 and 1.
- Can view as a series of switches that are either off (0) or on (1).
- 4-bit number uses 4 binary digits and ranges from 0000 or 0 to 1111 or  $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$
- Counting by 2's: 0 1 10 11 100

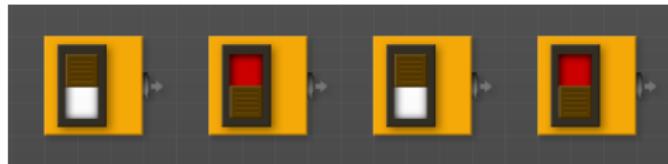
# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

- Only have two digits: 0 and 1.
- Can view as a series of switches that are either off (0) or on (1).
- 4-bit number uses 4 binary digits and ranges from 0000 or 0 to 1111 or  $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$
- Counting by 2's: 0 1 10 11 100 101

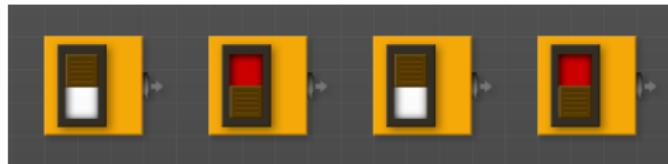
# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

- Only have two digits: 0 and 1.
- Can view as a series of switches that are either off (0) or on (1).
- 4-bit number uses 4 binary digits and ranges from 0000 or 0 to 1111 or  $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$
- Counting by 2's: 0 1 10 11 100 101 110

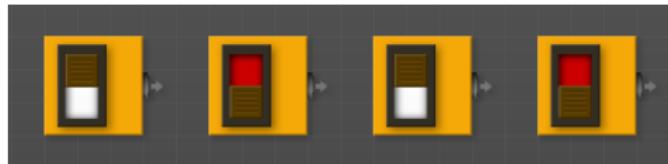
# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

- Only have two digits: 0 and 1.
- Can view as a series of switches that are either off (0) or on (1).
- 4-bit number uses 4 binary digits and ranges from 0000 or 0 to 1111 or  $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$
- Counting by 2's: 0 1 10 11 100 101 110 111...

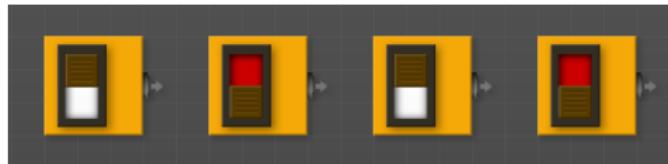
# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

- Only have two digits: 0 and 1.
- Can view as a series of switches that are either off (0) or on (1).
- 4-bit number uses 4 binary digits and ranges from 0000 or 0 to 1111 or  $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$
- Counting by 2's: 0 1 10 11 100 101 110 111...
- At the lowest level, information (data, commands, programs, etc.) on most computers is stored in binary.

# Binary Numbers



$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \end{array}$$

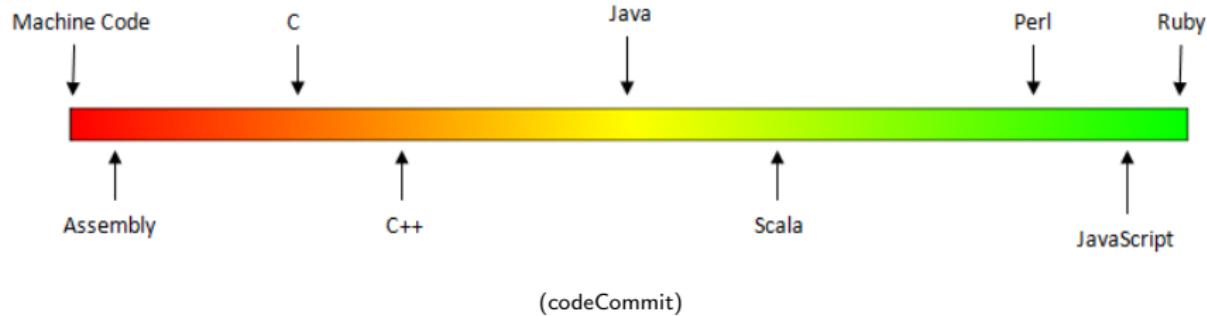
- Only have two digits: 0 and 1.
- Can view as a series of switches that are either off (0) or on (1).
- 4-bit number uses 4 binary digits and ranges from 0000 or 0 to 1111 or  $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$
- Counting by 2's: 0 1 10 11 100 101 110 111...
- At the lowest level, information (data, commands, programs, etc.) on most computers is stored in binary.
- Lecture slip: fill in the missing decimal, hex, and binary numbers.

# Today's Topics



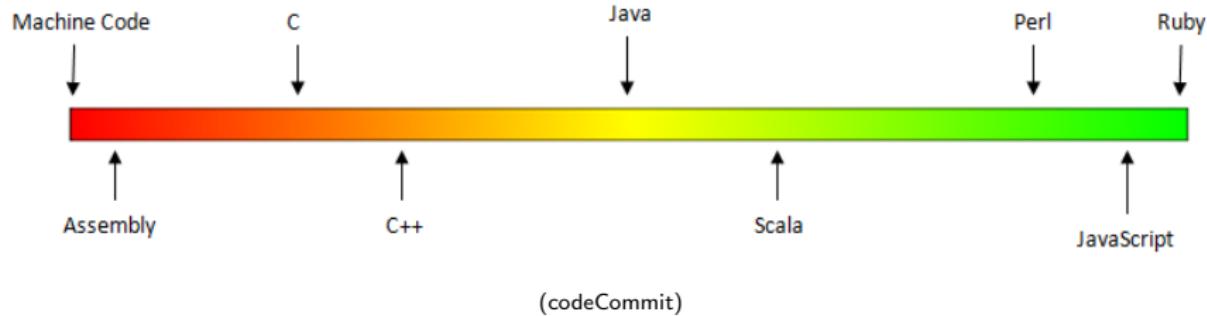
- Recap: folium and indefinite loops
- Design Patterns: Searching Data
- Data Representation
- **Machine Language**

# Low-Level vs. High-Level Languages



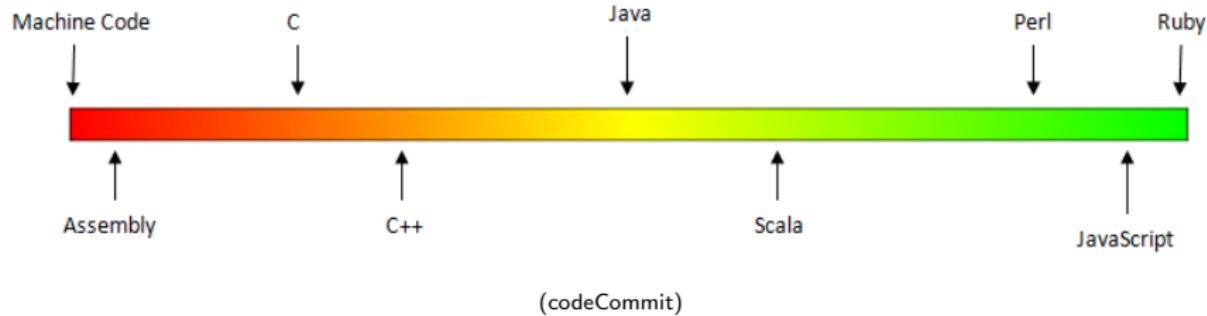
- Can view programming languages on a continuum.

# Low-Level vs. High-Level Languages



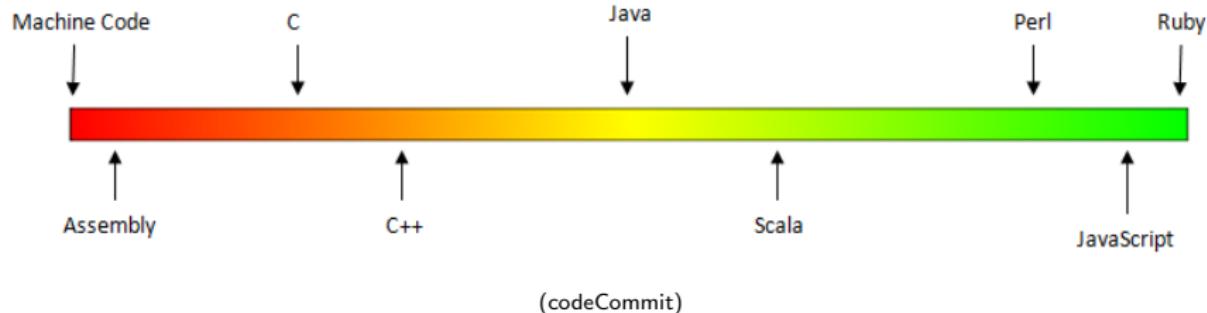
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

# Low-Level vs. High-Level Languages



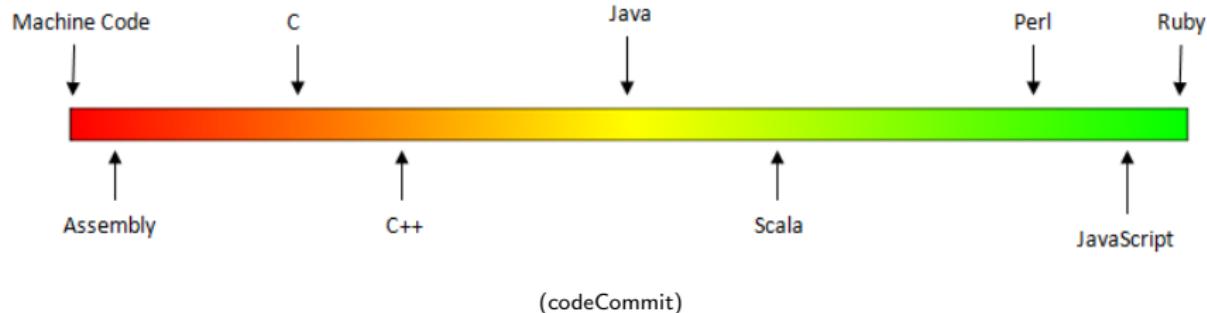
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

# Low-Level vs. High-Level Languages



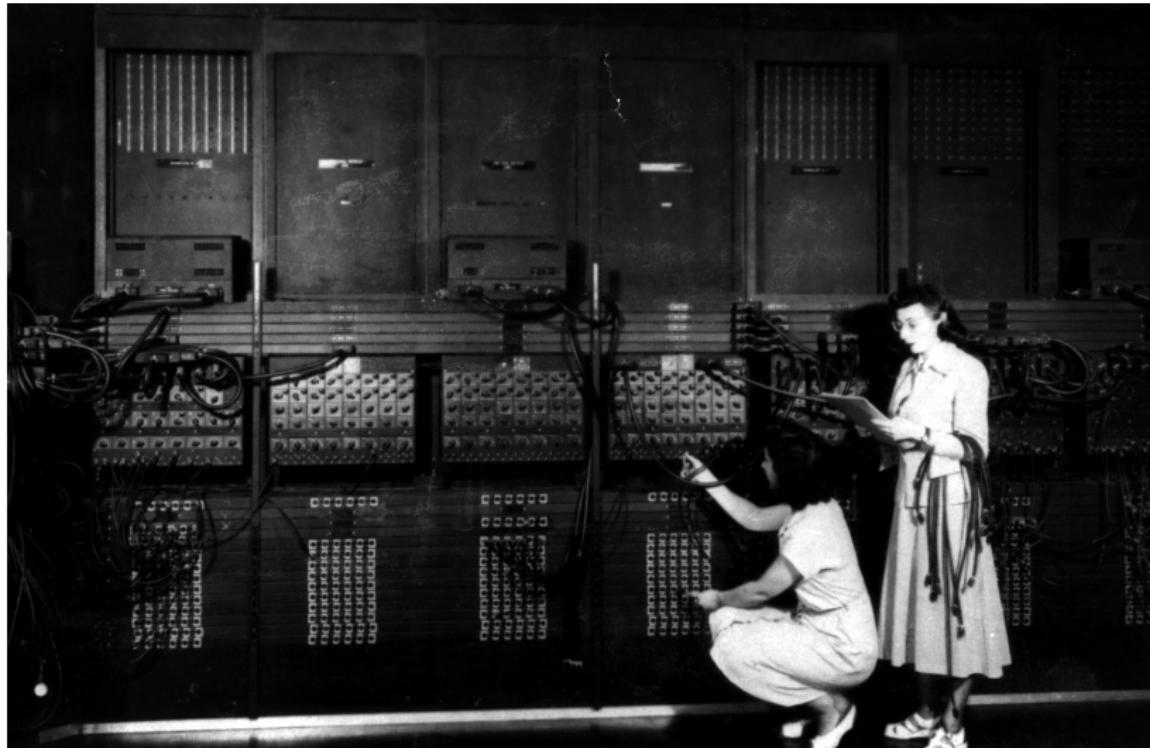
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
- Those that have strong abstraction (allow programming paradigms independent of the machine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

# Low-Level vs. High-Level Languages



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

# Machine Language



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

# Machine Language

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

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

BREAK

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

(wiki)

# Machine Language

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

(wiki)

# Machine Language

```

        ;B8 0000 C2 38 REP #30
        ;B8 0002 CLC SED
        ;B8 0004 F1 STX #1224
        ;B8 0006 34 32 LDH #1224
        ;B8 0007 60 21 45 ADD #1224
        ;B8 0008 0F B1 7F 01 STA #1224
        ;B8 000E D6 CLD
        ;B8 000F E2 38 SEP #38
        ;B8 0011 90 BYK
        A 2812

        ;B8 PC MM#012C A X Y SP DP IR
        ;B8 0012 B0100000 0000 0000 0002 CF7F 0000 0F
        & 2800

        BREAK

        ;B8 PC MM#012C A X Y SP DP IR
        ;B8 0013 B0110000 5555 0000 0002 CF7F 0000 0F
        n 7FFF 7195
        m 7FFF 7193
        l 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

(wiki)

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

# Machine Language



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

Assembly code:

```
R 002000 C2 3B REP #3B
R 002002 1B CLD
R 002003 FB SEI
R 002004 69 34 12 LDa $1234
R 002007 69 21 43 LDc $4321
R 002008 8F 03 7F 01 STA $017F03
R 00200E E9 30 CLD
R 002010 80 SER #38
R 002011 80 SWK
R 20012
```

Registers:

Reg	PC	MAR	D[31:24]	A	X	Y	SP	DP	R[31:24]	R[23:08]
PC	002000	00110000	0000 0000	0002	CFFF	0000 00				
MAR				5555	0000	0002				
D[31:24]				7FFF	2783					
A				55	55	00 00	00 00	00 00	00 00	00 00
X				00	00	00 00	00 00	00 00	00 00	00 00
Y				00	00	00 00	00 00	00 00	00 00	00 00
SP				00	00	00 00	00 00	00 00	00 00	00 00
DP				00	00	00 00	00 00	00 00	00 00	00 00

BREAK

(wiki)

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

# Machine Language



The screenshot shows a window titled 'WeMIPS' with assembly code and registers. The assembly code is:

```
R 002802 C2 3B REP #838  
R 002802 1B CLD  
R 002803 FB SEI  
R 002804 69 34 12 LDa #1234  
R 002807 69 21 43 LDc #4321  
R 002808 8F 03 7F 01 STa #17F03  
R 00280E E9 30 CLD  
R 002911 89 SEP #38  
R 2012 BREAK
```

The registers are:

PC	MAR	MDR	A	X	Y	SP	DP	RR
002802	00110000	0000 0000 0002	CFFF	0000 00				0000
002808	00110000	5555 0000 0002	CFFF	0000 00				0000
002911	00110000	55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00	CFFF	0000 00				0000

BREAK

(wiki)

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

# "Hello World!" in Simplified Machine Language

Line: 3 Go!

Show/Hide Demos

User Guide | Unit Tests | Docs

Addition Doubler Stav Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

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

Step Run  Enable auto switching

S T A V Stack Log

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

(WeMIPS)



# WeMIPS

User | 3 | Dat ShowHide Device

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

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

Debug

```
# Store 'Hello world!' at the top of the stack
1    ADDI $t0, $zero, 72 # $H
2    ADDI $t1, $zero, 101 # $e
3    ADDI $t2, $zero, 101 # $l
4    ADDI $t3, $zero, 108 # $o
5    ADDI $t4, $zero, 108 # $w
6    ADDI $t5, $zero, 108 # $r
7    ADDI $t6, $zero, 108 # $l
8    ADDI $t7, $zero, 108 # $d
9    ADDI $t8, $zero, 108 # $t
10   ADDI $t9, $zero, 108 # $e
11   ADDI $t10, $zero, 108 # $l
12   ADDI $t11, $zero, 108 # $o
13   ADDI $t12, $zero, 108 # $d # (epilogue)
14   ADDI $t13, $zero, 108 # $t
15   ADDI $t14, $zero, 108 # $e
16   ADDI $t15, $zero, 108 # $l
17   ADDI $t16, $zero, 108 # $o
18   ADDI $t17, $zero, 108 # $d
19   ADDI $t18, $zero, 108 # $t
20   ADDI $t19, $zero, 108 # $e
21   ADDI $t20, $zero, 108 # $l
22   ADDI $t21, $zero, 108 # $o
23   ADDI $t22, $zero, 108 # $d
24   ADDI $t23, $zero, 108 # $t
25   ADDI $t24, $zero, 108 # $e
26   ADDI $t25, $zero, 108 # $l
27   ADDI $t26, $zero, 108 # $o
28   ADDI $t27, $zero, 108 # $d # null
29   ADDI $t28, $zero, 108 # $t
30   ADDI $t29, $zero, 108 # $e
31   ADDI $t30, $zero, 4 # 4 is for print string
32   ADDI $t31, $zero, 0 # point to the log
33   syscall
```

Step	Run	<input checked="" type="checkbox"/> Enable auto switching			
S	T	A	V	Stack	Log
s0:	10				
s1:	9				
s2:	8				
s3:	22				
s4:	695				
s5:	976				
s6:	977				
s7:	419				

(Demo with WeMIPS)

# MIPS Commands

The screenshot shows a MIPS assembly debugger interface. At the top, there's a menu bar with 'File', 'Edit', 'Run', 'Help', 'Show/Hide Demo', 'User Guide', 'Unit Tests', and 'Docs'. Below the menu are tabs for 'Addition Counter', 'Btav', 'Looper', 'Stack Test', 'Hello World', 'Code Gen Save String', 'Interactive', 'Binary/Decimal', 'Decimal/Binary', and 'Debug'. The 'Debug' tab is selected.

The main area contains assembly code:

```
1 # Shows 'Hello world' at the top of the stack
2 .text
3 .globl _start
4 _start:
5    li $v0, 4
6    la $a0, _msg
7    syscall
8    li $v0, 1
9    li $a0, 10
10   syscall
11   li $v0, 4
12   la $a0, _msg
13   syscall
14   li $v0, 1
15   li $a0, 9
16   syscall
17   li $v0, 4
18   la $a0, _msg
19   syscall
20   li $v0, 1
21   li $a0, 10
22   syscall
23   li $v0, 4
24   la $a0, _msg
25   syscall
26   li $v0, 1
27   li $a0, 0
28   syscall
29   li $v0, 4
30   la $a0, _msg
31   syscall
32   li $v0, 4
33   la $a0, _log
34   syscall
```

To the right of the code, there's a register table:

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

Below the register table is a stack dump:

10	9	22	60	61	807
11	10	23	62	62	808
12	9	24	63	63	809
13	22	25	64	64	810
14	60	26	65	65	811
15	61	27	66	66	812
16	807	28	67	67	813
17	418	29	68	68	814

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

# MIPS Commands

The screenshot shows a software interface for debugging MIPS assembly code. At the top, there are tabs for 'User', 'File', and 'Run'. Below them are buttons for 'ShowFrame Demo', 'Addition Counter', 'Stack', 'Loop', 'Stack Test', 'Hello World', 'Interactive', 'Binary/Decimal', 'Decimal/Binary', and 'Debug'. To the right of these buttons are links for 'User Guide' and 'Unit Tests'. A 'Close' button is at the top right.

The main area contains assembly code:

```
1 # Shows "Hello world" at the top of the stack
2 .text
3 .globl _start
4 _start:
5    addiu   $sp,$sp,-16      # allocate space for arguments
6    addiu   $t0,$zero,101     # $t0 = 101
7    addiu   $t0,$t0,1          # $t0 = 102
8    addiu   $t0,$t0,1          # $t0 = 103
9    addiu   $t0,$t0,1          # $t0 = 104
10   addiu   $t0,$t0,1          # $t0 = 105
11   addiu   $t0,$t0,1          # $t0 = 106
12   addiu   $t0,$t0,1          # $t0 = 107
13   addiu   $t0,$t0,1          # $t0 = 108
14   addiu   $t0,$t0,1          # $t0 = 109
15   addiu   $t0,$t0,1          # $t0 = 110
16   addiu   $t0,$t0,1          # $t0 = 111
17   addiu   $t0,$t0,1          # $t0 = 112
18   addiu   $t0,$t0,1          # $t0 = 113
19   addiu   $t0,$t0,1          # $t0 = 114
20   addiu   $t0,$t0,1          # $t0 = 115
21   addiu   $t0,$t0,1          # $t0 = 116
22   addiu   $t0,$t0,1          # $t0 = 117
23   addiu   $t0,$t0,1          # $t0 = 118
24   addiu   $t0,$t0,1          # $t0 = 119
25   addiu   $t0,$t0,1          # $t0 = 120
26   addiu   $t0,$t0,1          # $t0 = 121
27   addiu   $t0,$t0,1          # $t0 = 122
28   addiu   $t0,$t0,1          # $t0 = 123
29   addiu   $t0,$t0,1          # $t0 = 124
30   addiu   $t0,$t0,1          # $t0 = 125
31   addiu   $t0,$t0,1          # $t0 = 126
32   addiu   $t0,$t0,4          # $t0 = 127 (for print string)
33   li      $v0,4              # print to the log
34   syscall
```

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

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

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

# MIPS Commands

The screenshot shows the ShowFide Demo interface. At the top, there are tabs for User, ShowFide Demo, and other options like Addition, Counter, Stack Test, Interactive, etc. Below the tabs is a menu bar with File, Edit, Tools, Options, Help, and Exit. A toolbar below the menu has buttons for Addition, Counter, Stack Test, Interactive, Code Gen, Save String, Interactive, Binary, Decimal, and Hexadecimal. A Debug button is also present. The main area contains assembly code and a register dump.

Assembly code:

```
1 # Shows "Hello world!" at the top of the stack
2 .text
3 .globl _start
4 _start:
5    li $t0, 0x48454c4c # Hello
6    li $t1, 0x6f6f6f6f # world
7    add $t2, $t0, $t1      # $t2 = Hello + world
8    add $t3, $t2, $t2      # $t3 = $t2 + $t2 = Hello + world + Hello + world
9    add $t4, $t3, $t3      # $t4 = $t3 + $t3 = Hello + world + Hello + world + Hello + world
10   add $t5, $t4, $t4      # $t5 = $t4 + $t4 = Hello + world + Hello + world + Hello + world + Hello + world
11   add $t6, $t5, $t5      # $t6 = $t5 + $t5 = Hello + world + Hello + world + Hello + world + Hello + world + Hello + world
12   add $t7, $t6, $t6      # $t7 = $t6 + $t6 = Hello + world + Hello + world
13   add $t8, $t7, $t7      # $t8 = $t7 + $t7 = Hello + world + Hello + world
14   sb $t8, 4($t0)        # print to memory
15   li $t9, 0x0            # zero
16   sb $t9, 7($t0)        # print to memory
17   li $t10, 0x48          # Newline
18   sb $t10, 8($t0)       # print to memory
19   li $t11, 0x0            # zero
20   sb $t11, 9($t0)       # print to memory
21   li $t12, 0x48          # Newline
22   sb $t12, 10($t0)      # print to memory
23   add $t13, $t0, $t0      # $t13 = $t0 + $t0 = 0x48454c4c + 0x48454c4c = 0x968a968a
24   add $t14, $t13, $t13    # $t14 = $t13 + $t13 = 0x968a968a + 0x968a968a = 0x1e311e31
25   add $t15, $t14, $t14    # $t15 = $t14 + $t14 = 0x1e311e31 + 0x1e311e31 = 0x3e623e62
26   add $t16, $t15, $t15    # $t16 = $t15 + $t15 = 0x3e623e62 + 0x3e623e62 = 0x7e247e24
27   add $t17, $t16, $t16    # $t17 = $t16 + $t16 = 0x7e247e24 + 0x7e247e24 = 0xe049e049
28   add $t18, $t17, $t17    # $t18 = $t17 + $t17 = 0xe049e049 + 0xe049e049 = 0x1e091e09
29   add $t19, $t18, $t18    # $t19 = $t18 + $t18 = 0x1e091e09 + 0x1e091e09 = 0x3e183e18
30   add $t20, $t19, $t19    # $t20 = $t19 + $t19 = 0x3e183e18 + 0x3e183e18 = 0x7e367e36
31   add $t21, $t20, $t20    # $t21 = $t20 + $t20 = 0x7e367e36 + 0x7e367e36 = 0xe073e073
32   syscall               # print to the log
```

Register dump:

S	T	A	V	Stack	Log
\$0				10	
\$1				9	
\$2				8	
\$3				7	
\$4				6	
\$5				5	
\$6				4	
\$7				3	
\$8				2	
\$9				1	
\$10				0	
\$11				418	

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

# MIPS Commands

The screenshot shows a MIPS assembly debugger interface. At the top, there's a menu bar with tabs like User, File, Go, Show/Hide Demo, Addition, Subtraction, Bitwise, Looper, Stack Test, Hello World, Code Gen Save String, Interactive, Binary/Decimal, Decimal/Binary, and Debug. Below the menu is a toolbar with icons for Run, Stop, Break, and others. The main area has tabs for Step, Run, and Log. The Step tab is active, showing assembly code:

```
1 # Shows "Hello world" at the top of the stack
2 .text
3 .globl _start
4 _start:
5    addi   $t0, $zero, 100 # $t0 = 100
6    addi   $t1, $zero, 101 # $t1 = 101
7    addi   $t2, $zero, 100 # $t2 = 100
8    addi   $t3, $zero, 100 # $t3 = 100
9    addi   $t4, $zero, 100 # $t4 = 100
10   addi   $t5, $zero, 100 # $t5 = 100
11   addi   $t6, $zero, 100 # $t6 = 100
12   addi   $t7, $zero, 100 # $t7 = 100
13   addi   $t8, $zero, 100 # $t8 = 100
14   addi   $t9, $zero, 100 # $t9 = 100
15   addi   $t10, $zero, 100 # $t10 = 100
16   addi   $t11, $zero, 100 # $t11 = 100
17   addi   $t12, $zero, 100 # $t12 = 100
18   addi   $t13, $zero, 100 # $t13 = 100
19   addi   $t14, $zero, 100 # $t14 = 100
20   addi   $t15, $zero, 100 # $t15 = 100
21   addi   $t16, $zero, 100 # $t16 = 100
22   addi   $t17, $zero, 100 # $t17 = 100
23   addi   $t18, $zero, 100 # $t18 = 100
24   addi   $t19, $zero, 100 # $t19 = 100
25   addi   $t20, $zero, 100 # $t20 = 100
26   addi   $t21, $zero, 100 # $t21 = 100
27   addi   $t22, $zero, 100 # $t22 = 100
28   addi   $t23, $zero, 100 # $t23 = 100
29   addi   $t24, $zero, 100 # $t24 = 100
30   addi   $t25, $zero, 100 # $t25 = 100
31   addi   $t26, $zero, 100 # $t26 = 100
32   addi   $t27, $zero, 100 # $t27 = 100
```

Below the assembly code is a log window showing register values:

S	T	A	V	Stack	Log
s0	10				
s1	9				
s2	8				
s3	7				
s4	6				
s5	5				
s6	407				
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

The screenshot shows the ShowMIPS Demo application interface. At the top, there are tabs for User, ShowMIPS Demo, and Run. Below the tabs are buttons for Addition, Counter, IfElse, Looper, StackTest, Hello World, CodeGen, SaveString, Interactive, BinaryDec, DecimalDec, and DecimalBin. A Debug button is also present. On the right, there are links for User Guide and Unit Tests.

The main area displays assembly code and a register dump. The assembly code is as follows:

```
# Shows "Hello world" at the top of the stack
1    .text
2    .globl _start
3    .type _start, @function
4    _start:
5        addi $t0, $zero, 111 # a
6        addi $t1, $zero, 110 # b
7        addi $t2, $zero, 110 # c
8        addi $t3, $zero, 110 # d
9        addi $t4, $zero, 110 # e
10       addi $t5, $zero, 110 # f
11       addi $t6, $zero, 110 # g
12       addi $t7, $zero, 110 # h
13       addi $t8, $zero, 110 # i
14       addi $t9, $zero, 110 # j
15       addi $t10, $zero, 110 # k
16       addi $t11, $zero, 110 # l
17       addi $t12, $zero, 110 # m
18       addi $t13, $zero, 110 # n
19       addi $t14, $zero, 110 # o
20       addi $t15, $zero, 110 # p
21       addi $t16, $zero, 110 # q
22       addi $t17, $zero, 110 # r
23       addi $t18, $zero, 110 # s
24       addi $t19, $zero, 110 # t
25       addi $t20, $zero, 110 # u
26       addi $t21, $zero, 110 # v
27       addi $t22, $zero, 0 # null
28       addi $t23, $zero, 4 # 4 is for print string
29       addi $t24, $zero, 5 # print to the log
30       syscall
```

On the right, a register dump table shows the values of registers \$s0 through \$t24. The table has columns for S, T, A, V, Stack, and Log.

S	T	A	V	Stack	Log
\$0				10	
\$1				9	
\$2				8	
\$3				7	
\$4				6	
\$5				5	
\$6				4	
\$7				3	
\$8				2	
\$9				1	
\$10				0	
\$11				418	
\$12					
\$13					
\$14					
\$15					
\$16					
\$17					
\$18					
\$19					
\$20					
\$21					
\$22					
\$23					
\$24					

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

# MIPS Commands

The screenshot shows a window titled "ShowMIPS Demo". At the top, there are tabs for "Assembly", "Decimal", "Binary", "Hex", "Interactive", and "Debug". Below these are buttons for "Code Gen", "Save String", "Interactive", "Decimal", "Binary", and "Hex". A "Debug" button is also present. The main area contains assembly code and a register dump.

Assembly code:

```
1 # Shows "Hello world!" at the top of the stack
2 .text
3 .globl _start
4 _start:
5    addi   $s0, $zero, 101 # $s0 = 101
6    addi   $s1, $zero, 102 # $s1 = 102
7    addi   $s2, $zero, 103 # $s2 = 103
8    addi   $s3, $zero, 104 # $s3 = 104
9    addi   $s4, $zero, 105 # $s4 = 105
10   addi   $s5, $zero, 106 # $s5 = 106
11   addi   $s6, $zero, 107 # $s6 = 107
12   addi   $s7, $zero, 108 # $s7 = 108
13   addi   $s8, $zero, 109 # $s8 = 109
14   addi   $s9, $zero, 110 # $s9 = 110
15   addi   $s10, $zero, 111 # $s10 = 111
16   addi   $s11, $zero, 112 # $s11 = 112
17   addi   $s12, $zero, 113 # $s12 = 113
18   addi   $s13, $zero, 114 # $s13 = 114
19   addi   $s14, $zero, 115 # $s14 = 115
20   addi   $s15, $zero, 116 # $s15 = 116
21   addi   $s16, $zero, 117 # $s16 = 117
22   addi   $s17, $zero, 118 # $s17 = 118
23   addi   $s18, $zero, 119 # $s18 = 119
24   addi   $s19, $zero, 120 # $s19 = 120
25   addi   $s20, $zero, 121 # $s20 = 121
26   addi   $s21, $zero, 122 # $s21 = 122
27   addi   $s22, $zero, 0 # (null)
28
29   addi   $s0, $zero, 4 # $s0 = 4 for print string
30   addi   $s0, $zero, 5 # $s0 = 5 for syscall
31
32   syscall
```

Register dump:

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

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

# MIPS Commands

The screenshot shows the StackFrame Demo application window. At the top, there are tabs for 'User', '32', '64', 'ShowFrame Demo', 'User Guide', 'Unit Tests', and 'Doc'. Below the tabs are several command buttons: 'Addition', 'Double', 'Btav', 'Loop', 'Stack Test', 'Hello World', 'Code Gen', 'Save String', 'Interactive', 'Binary', 'Decimal', 'Hexadecimal', and 'Debug'. The main area displays assembly code for a 'Hello World' program, starting with a stack frame setup and followed by the standard main loop. To the right of the code is a register dump table with columns for \$, T, A, V, Stack, and Log. The registers shown are \$0 through \$31, with their current values listed.

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

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

# MIPS Commands

The screenshot shows the ShowMe Demo interface. At the top, there are tabs for User, ShowMe Demo, Addition, Subtraction, Bitwise, Looper, Stack Test, Hello World, Code Gen, Save String, Interactive, Binary, Decimal, and Debug. Below the tabs is a text area containing MIPS assembly code. To the right is a register dump window titled 'Registers' with columns for S, T, A, V, Stack, and Log.

```
# Shows "Hello world" at the top of the stack
1 L:    .text
2      .globl _start
3      .type _start, @function
4      _start:
5      addi   $t0, $zero, 101 # $t0 = 101
6      addi   $t1, $zero, 100 # $t1 = 100
7      addi   $t2, $zero, 100 # $t2 = 100
8      addi   $t3, $zero, 100 # $t3 = 100
9      addi   $t4, $zero, 100 # $t4 = 100
10     addi   $t5, $zero, 100 # $t5 = 100
11     addi   $t6, $zero, 100 # $t6 = 100
12     addi   $t7, $zero, 100 # $t7 = 100
13     addi   $t8, $zero, 100 # $t8 = 100
14     addi   $t9, $zero, 100 # $t9 = 100
15     addi   $t10, $zero, 100 # $t10 = 100
16     addi   $t11, $zero, 100 # $t11 = 100
17     addi   $t12, $zero, 100 # $t12 = 100
18     addi   $t13, $zero, 100 # $t13 = 100
19     addi   $t14, $zero, 100 # $t14 = 100
20     addi   $t15, $zero, 100 # $t15 = 100
21     addi   $t16, $zero, 100 # $t16 = 100
22     addi   $t17, $zero, 100 # $t17 = 100
23     addi   $t18, $zero, 100 # $t18 = 100
24     addi   $t19, $zero, 100 # $t19 = 100
25     addi   $t20, $zero, 100 # $t20 = 100
26     addi   $t21, $zero, 100 # $t21 = 100
27     addi   $t22, $zero, 0 # ($t22 = 0)
28     addi   $t23, $zero, 0 # ($t23 = 0)
29     addi   $t24, $zero, 0 # ($t24 = 0)
30     addi   $t25, $zero, 0 # ($t25 = 0)
31     addi   $t26, $zero, 0 # ($t26 = 0)
32     addi   $t27, $zero, 0 # ($t27 = 0)
33     addi   $t28, $zero, 0 # ($t28 = 0)
34     addi   $t29, $zero, 0 # ($t29 = 0)
35     addi   $t30, $zero, 0 # ($t30 = 0)
36     addi   $t31, $zero, 0 # ($t31 = 0)
37
38     li     $t0, 65
39     li     $t1, 69
40     li     $t2, 76
41     li     $t3, 73
42     li     $t4, 77
43     li     $t5, 72
44     li     $t6, 70
45     li     $t7, 74
46     li     $t8, 71
47     li     $t9, 75
48     li     $t10, 78
49     li     $t11, 79
50     li     $t12, 80
51     li     $t13, 81
52     li     $t14, 82
53     li     $t15, 83
54     li     $t16, 84
55     li     $t17, 85
56     li     $t18, 86
57     li     $t19, 87
58     li     $t20, 88
59     li     $t21, 89
60     li     $t22, 90
61     li     $t23, 91
62     li     $t24, 92
63     li     $t25, 93
64     li     $t26, 94
65     li     $t27, 95
66     li     $t28, 96
67     li     $t29, 97
68     li     $t30, 98
69     li     $t31, 99
70
71     addi   $t0, $t0, 104 # for print string
72     addi   $t0, $t0, 104 # print to the log
73
74     syscall
```

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

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

# MIPS Commands

The screenshot shows the StackFrame Demo application interface. At the top, there are tabs for User, ShowFrame Demo, Addition Demo, ItInv, Looper, Stack Test, Hello World, Code Gen Save String, Interactive, Direct Decimal, Decimal Binary, and Debug. Below the tabs is a text area containing assembly code for a 'Hello World' program. The code includes instructions like ADDI, ADD, SUB, and LW, along with comments and labels. To the right of the code is a table titled 'Registers' with columns S, T, A, V, Stack, and Log. The registers listed are \$0 through \$31, with their current values displayed in the S column.

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

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

# In Pairs or Triples:

Line: 3 Go! Show/Hide Demos

User Guide | Unit Tests | Docs

Addition Doubler Stav Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

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

Step Run  Enable auto switching

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

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

# WeMIPS

The screenshot shows the WeMIPS IDE interface. At the top, there are tabs for "Live", "3", "Data", and "ShowHide Device". Below these are navigation links: "Addition Doubler", "Stax", "Looper", "Stack Test", and "Hello World". Further down are "Code Gen Save String", "Interactive", "Binary2 Decimal", and "Decimal2 Binary". A "Debug" button is also present.

The main area displays assembly code:

```
# Store 'Hello world!' at the top of the stack
    .text
    .globl _start
_start:
    # move $0, $zero, $2 # N
    ADDI $t0, $zero, 72 # N
    # move $1, $zero, $4 # E
    ADDI $t1, $zero, 101 # E
    # move $2, $zero, $1 # S
    ADDI $t2, $zero, 108 # S
    # move $3, $zero, $0 # O
    ADDI $t3, $zero, 109 # O
    # move $4, $zero, $3 # (space)
    ADDI $t4, $zero, 32 # (space)
    # move $5, $zero, $5 # g
    ADDI $t5, $zero, 117 # g
    # move $6, $zero, $6 # e
    ADDI $t6, $zero, 101 # e
    # move $7, $zero, $11 # l
    ADDI $t7, $zero, 108 # l
    # move $8, $zero, $13 # o
    ADDI $t8, $zero, 111 # o
    # move $9, $zero, $14 # d
    ADDI $t9, $zero, 109 # d
    # move $10, $zero, $10 # r
    ADDI $t10, $zero, 105 # r
    # move $11, $zero, $33 # i
    ADDI $t11, $zero, 51 # i
    # move $12, $zero, $0 # null
    ADDI $t12, $zero, 0 # null
    # move $13, $zero, $10 # l
    ADDI $t13, $zero, 10 # l
    # move $14, $zero, $4 # 4 is for print string
    ADDI $t14, $zero, 4 # 4
    # move $15, $zero, $0 # point to the log
    ADDI $t15, $zero, 0 # point to the log
    # syscall
    SYSCALL
```

Step	Run	<input checked="" type="checkbox"/> Enable auto switching			
S	T	A	V	Stack	Log
s0:	10				
s1:	9				
s2:	8				
s3:	22				
s4:	695				
s5:	576				
s6:	327				
s7:	419				

(Demo with WeMIPS)

# Recap

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



# Recap

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



# Recap



- On lecture slip, write down a topic you wish we had spent more time (and why).
- Searching through data is a common task— built-in functions and standard design patterns for this.
- We use different base numbers (i.e. binary and hexadecimal) to represent data.

# Recap



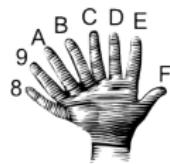
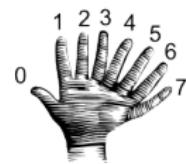
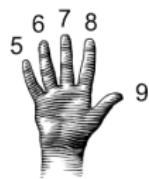
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Searching through data is a common task— built-in functions and standard design patterns for this.
- We use different base numbers (i.e. binary and hexadecimal) to represent data.
- Programming languages can be classified by the level of abstraction and direct access to data.

# Recap



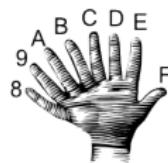
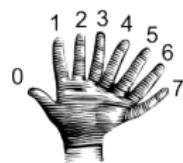
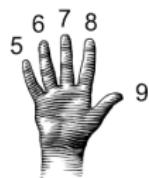
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Searching through data is a common task— built-in functions and standard design patterns for this.
- We use different base numbers (i.e. binary and hexadecimal) to represent data.
- Programming languages can be classified by the level of abstraction and direct access to data.
- Pass your lecture slips to the aisles for the UTAs to collect.

# Practice Quiz & Final Questions



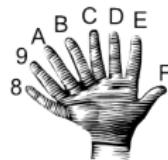
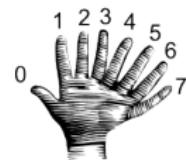
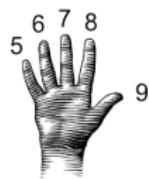
- Lightning rounds:

# Practice Quiz & Final Questions



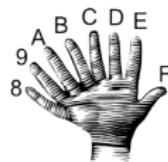
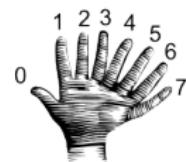
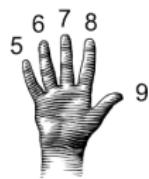
- Lightning rounds:
  - ▶ write as much you can for 60 seconds;

# Practice Quiz & Final Questions



- Lightning rounds:
  - ▶ write as much you can for 60 seconds;
  - ▶ followed by answer; and

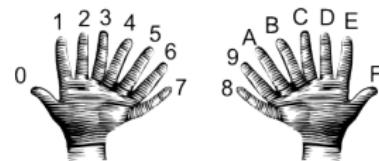
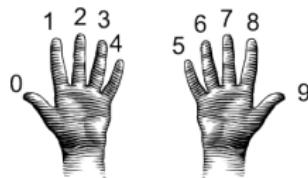
# Practice Quiz & Final Questions



- Lightning rounds:

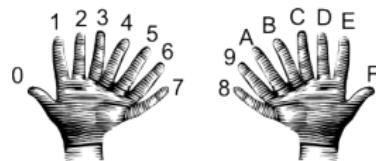
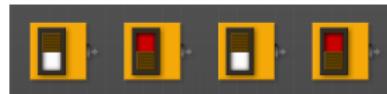
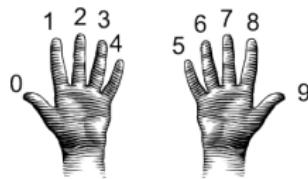
- ▶ write as much you can for 60 seconds;
- ▶ followed by answer; and
- ▶ repeat.

# Practice Quiz & Final Questions



- Lightning rounds:
  - ▶ write as much you can for 60 seconds;
  - ▶ followed by answer; and
  - ▶ repeat.
- Past exams are on the webpage ([under Final Exam Information](#)).

# Practice Quiz & Final Questions



- Lightning rounds:
  - ▶ write as much you can for 60 seconds;
  - ▶ followed by answer; and
  - ▶ repeat.
- Past exams are on the webpage ([under Final Exam Information](#)).
- Theme: Data Representation! Starting with F17, Mock, #2 and #3.