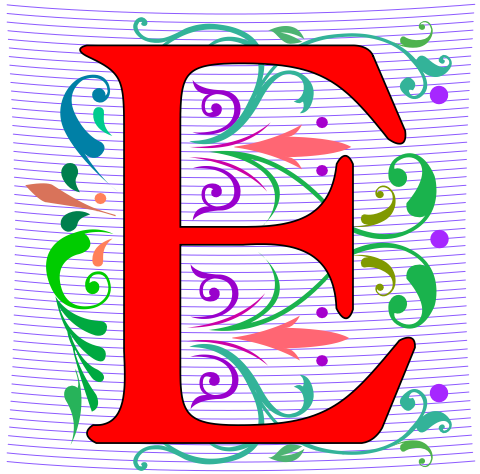
CompSci 101
Conditionals (Cont'd), Collections, Strings, Lists

Announcements

- Upcoming due dates
  - All Sakai quizzes due @145pm day of lecture
  - Assignment 1-due 2/11 @1130pm
- Piazza channel
  - Direct questions here

`E` is for …

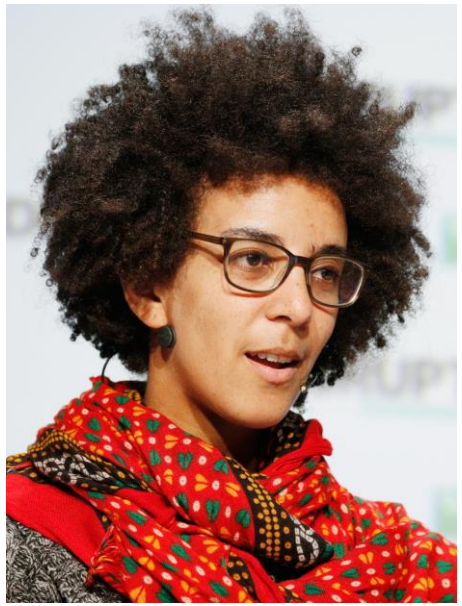
- Escape Sequence
  - Why `\n` is newline and `\t` is a tab
- Encryption
  - From Caesar Ciphers to SSL and beyond
- Enumerate
  - Adding counters to iterable
- Emoticon
  -



Computer Scientists to Know

- Dr. Clarence “Skip” Ellis
  - 1st Black person to earn a Ph.D. in computer science

- Dr. Timnit Gebru
  - Co-founder, Black in AI
  - Ethical AI researcher



PFTD

- Selections/Conditionals continued
- Strings
- Lists

- “The mere imparting of information is not education.”
  - Dr. Carter G. Woodson

Selection/Conditionals: if…elif…else

```
if BOOLEAN_CONDITION:
    CODE_BLOCK_A
```

```
if BOOLEAN_CONDITION:
    CODE_BLOCK_A
else:
    CODE_BLOCK_B
```

```
if BOOLEAN_CONDITION:
    CODE_BLOCK_A
elif BOOLEAN_CONDITION:
    CODE_BLOCK_B
else:
    CODE_BLOCK_C
```

```
if __name__ == '__main__':
    num1 = 5

    if num1 == 5:
        print("The number is 5!")
    else:
        if num1 < 5:
            print("The number is less than 5!")
        else:
            print("The number is greater than 5!")
```

```
if __name__ == '__main__':
    num1 = 2

    if num1 == 5:
        print("The number is 5!")
    elif num1 < 5:
        print("The number is less than 5!")
    else:
        print("The number is greater than 5!")
```

WOTO 4: Review

- Extending your program from WOTO 3
- Simulate rolling two dice
  - “Roll” two dice
  - Results of the rolls sent to function named *sum*
    - Sum dice values
    - If sum is 7 or 11, then output “You win!”
    - Otherwise, output “Next time!”

Functions Calling Other Functions

```
def function1(parameter):
        …
        result=function2(parameter2)
        return result

def function2(parameter2):
        …
        return result2

 if _ _ name _ _ = = ‘_ _main_ _’
        output=function1(argument)
        print(output)
```

Example code(PyCharm)

Collection Data Type

- Collection of books, toys, shoes
  - Direct access to each item
- Comprised of smaller pieces
  - Strings and lists
- Strings
  - Smaller strings of size one char
  - Empty string- “” or ‘’
- Operations on strings
  - + →concatenation
  - * →repetition

```
if __name__ == '__main__':
    result1 = "Hey there!"
    result2 = "How are you?"

    # concatenate two strings
    result = result1 + result2
    print(result)
```

```
if __name__ == '__main__':
    result1 = "Hey there!"
    result2 = "How are you?"

    # repeat a string
    result = result1 * 3
    print(result)
```

Indexing a String

string_name[*index*]

- Index: 0 to (string_length-1)
- **Whitespaces count**

```
if __name__ == '__main__':
    result1 = "Hey there!"
    result2 = "How are you?"

    # get lengths of strings
    print(len(result1))
    print(len(result2))
```

```
if __name__ == '__main__':
    result1 = "Hey there!"
    result2 = "How are you?"

    # get lengths of strings
    print(len(result1))
    print(len(result2))

    print(result1[0])
    print(result2[5])
    print(result1[-1])
    print(result2[-3])
```

Slicing Strings

- Slicing bread, tomatoes, etc.
- Substring (smaller part) of the larger string

string_name[*n*:*m*]

```
if __name__ == '__main__':
    result1 = "Hey there!"
    result2 = "How are you?"

    # slice strings
    print(result1[2:5])
    print(result2[4:8])
```

```
if __name__ == '__main__':
    result1 = "Hey there!"
    result2 = "How are you?"

    # slice strings
    print(result1[:5])
    print(result2[4:])
```

WOTO 1: http://bit.ly/101s21-0204-1b

Comparing Strings

- Compares strings to determine the relationship between them
  - ==, >, <, >=, <=, !=

- *string1* == *string2*

need to output this or store the result

```
if __name__ == '__main__':
    result1 = "Hey there!"
    result2 = "How are you?"

    # compare strings
    print(result1 == result2)
    print(result1 != result2)
    print(result1 > result2)
    print(result1 < result2)
```

in and *not in* operators

- Is string1 a substring of string2?

string1 in *string2*

string can be a variable or a string literal (e.g., “This is literally an example of a string literal.”)

```
if __name__ == '__main__':
    result1 = "Hey there!"
    result2 = "How are you?"

    # check in/not in tests
    print(result1 in result2)
    print(result1 not in result2)
    print(result1 in result1)

    print("Hey" in "Hey Ya!")
    print("" in "Hey Ya!")
    print("Hey Ya!" not in "Hey Ya!")
```

WOTO 2: http://bit.ly/101s21-0204-2

Lists

- Groceries, errands, names, etc.
- Lists are:
  - Ordered
  - Mutable
  - Duplicate elements allowed
  - Elements don’t have to be the same type

list_name=[*item1*, *item2, …item6*]

****only top-level items in list****

```
if __name__ == '__main__':
    ages = [12, 44, 10, 21]
    names = ["Kim", "Janay", "TJ", "Nia"]
    combo = ["Tim", 13, "Ashanti", [40, "Pink"]]

    # output lists
    print(ages)
    print(names)
    print(combo)
```

List access and length

- Similar to strings

list_name[*index*]

- list_name-your variable name
- index-character element directly accessing
  - leftmost 0 to list_length-1

- What about list_name[-1]?

```
if __name__ == '__main__':
    ages = [12, 44, 10, 21]
    names = ["Kim", "Janay", "TJ", "Nia"]
    combo = ["Tim", 13, "Ashanti", [40, "Pink"]]

    # print list length
    print(len(ages))
    print(len(names))
    print(len(combo))

    # directly access elements
    print(ages[1])
    print(names[3])
    print(combo[-1])
```

Slicing Lists

- Sublist (smaller part) of the larger list

list_name[*n*:*m*]

n-index of the first character in the sublist

m-index of the character that **immediately follows the last character in the sublist**

```
if __name__ == '__main__':
    ages = [12, 44, 10, 21]
    names = ["Kim", "Janay", "TJ", "Nia"]
    combo = ["Tim", 13, "Ashanti", [40, "Pink"]]

    # slice lists
    print(ages[1:3])
    print(names[:2])
    print(combo[1:])
```

in and *not in* operators

- Is list1 a member of list2?

list1 in *list2*

list1 not in *list2*

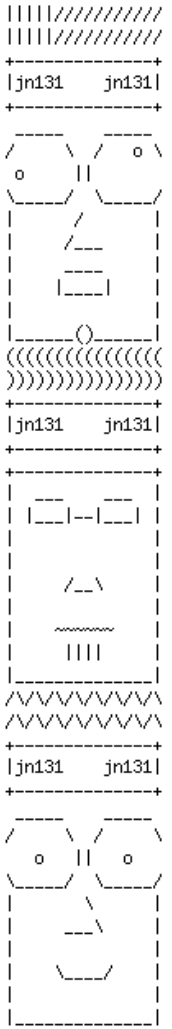
```
if __name__ == '__main__':
    ages = [12, 44, 10, 21]
    names = ["Kim", "Janay", "TJ", "Nia"]
    combo = ["Tim", 13, "Ashanti", [40, "Pink"]]

    # check membership
    print(21 in ages)
    print("13" not in combo)
    print("Pink" in combo)
```

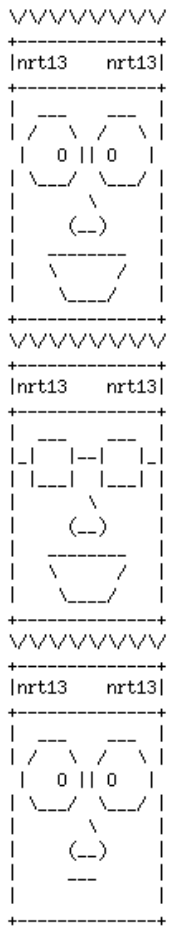
WOTO 3: http://bit.ly/101s21-0204-3

Assignment 1: Totem Poles

elf totem



selfie totem



self totem

Learning Goals: Totem Pole

- Understand differences and similarities:
  - Function definitions vs function calls
  - Functions with return statements vs those without
  - Functions with parameters vs those without
  - Functions can be arguments

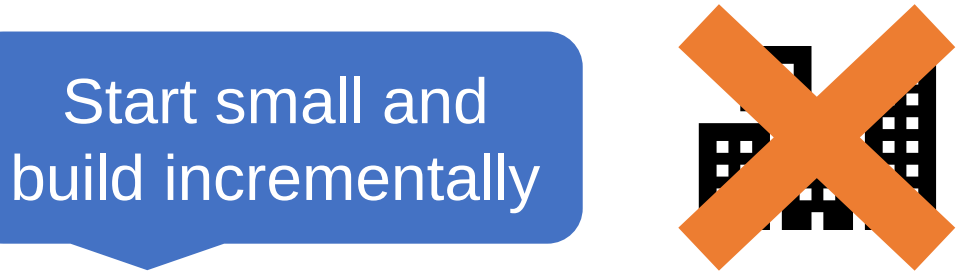
- Be creative and learn lesson(s) about software design and engineering
  - Create a small, working program, make incremental improvements.
  - Read the directions and understand specifications!

Function Name Format

Function	Parameters	Returns	Example
`part_DESCRIPTION`	No parameters	A string	`part_smiling_mouth`
`DESCRIPTION_head`	No parameters	No return value, only prints	`happy_head`
`head_with_DESCRIPTION`	1 or 2 parameters of type function	No return value, only prints	`head_with_mouth`
`totem_DESCRIPTION`	No parameters	No return value, calls head functions	`totem_fixed`, `totem_selfie`, `totem_random`
`selfie_band`, `head_random` – helper functions!			

Creating your program

Start small and build incrementally



With functions grow by…

```
7  def part_simple_hair():
8      a = r"0123456789012345678"
9      a = r" /\/\/\/\/\/\/\ "
10     return a
11
12 def happy_head():
13     print(part_simple_hair())
14
15 def totem_fixed():
16     happy_head()
17
18 def totem_selfie():
19     pass
20
21 def totem_random():
22     pass
23
24 if __name__ == '__main__':
25     print("\nfixed totem\n")
26     totem_fixed()
27
28     print("\nself totem\n")
29     totem_selfie()
30
31     print("\nrandom totem\n")
32     totem_random()
```

- Minimal code that does run and can be submitted
- Where go from here?
  - Add head part functions to create happy_head()
  - Create the next head function for totem_fixed and any new head part functions
  - Try a head_with function
  - Go to the next totem
  - etc.

Totem Assignment by Tuesday

- At minimum…
- Read the assignment
- Do the Totem reading quiz
- Create initial design
- Create project and start writing code (do not need to finish)

- Goal: Find your first question about how to do this assignment then ask on Piazza or at consulting/office hours

Remember

- Work smarter, not harder
- Design first
- Try to identify where you are stuck
  - Identify resources to help solve problem
- Leverage your design and PythonTutor to understand program flow of control
  - http://pythontutor.com