CSE 106

Lecture 3 – Python
Conditions, Loops, Functions, Input/Output

Acknowledgement: many code examples from learnpython.org

Conditions

- Python uses Boolean logic to evaluate conditions
- True and False are returned when an expression is compared or evaluated

```
x = 2
print(x == 2) # prints out True
print(x == 3) # prints out False
print(x < 3) # prints out True
```

Conditions - Boolean Operators

- The "and" and "or" Boolean operators allow building complex Boolean expressions

```
name = "John"
age = 23
if name == "John" and age == 23:
    print("Your name is John. You are 23 years old.")

if name == "John" or name == "Rick":
    print("Your name is either John or Rick.")
```

Conditions - The "in" operator

- The "in" operator can be used to check if a specified object exists within a list

```
name = "John"
nameList = ["John", "Rick"]
if name in nameList:
    print("Your name is either John or Rick.")
```

Conditions – If/Else

- Python uses indentation to define code block, just like the if/else statement below

```
x = 2
if x == 2:
    print("x equals two!")
    print(“heck yeah!”)
else:
    print("x does not equal to two.")
```

Conditions – Else If

- Else if statements (`elif`) let you have as many conditions as you like

```
is_cool = False
is_awesome = False

if is_cool is True:
    print("Cool!")
elif is_awesome is True:  # else if
    print("Awesome!")
else:
    print("Meh.")
```

Conditions – “is” Operator

- Unlike the double equals operator "==", the "is" operator does not match the values of the variables, but the instances themselves

```
x = [1,2,3]
y = [1,2,3]
print(x == y) # Prints out True
print(x is y) # Prints out False
```

Conditions – The "not" operator

- "not" before a Boolean expression inverts it (the opposite)

```
print(not False) # Prints out True
print((not False) == (False)) # Prints out False
```

Loops – For Loop

- For loops iterate over a given sequence
- They create a single value for each element in the sequence

```
primeList = [2, 3, 5, 7]
for prime in primeList:
    print(prime)
```

Loops – For Loop with Range

- For loops can iterate over a sequence of numbers using the “range”
- Range returns a sequence of numbers
  - range(stop)
  - range(start, stop, step (optional))

```
for x in range(5):  # Prints out the numbers 0,1,2,3,4
    print(x)
for x in range(3, 6):  # Prints out 3,4,5
    print(x)
for x in range(3, 8, 2):  # Prints out 3,5,7
    print(x)
```

Loops – Break Statement

- Break is used to exit a loop

```
# Prints out 0,1,2,3,4,5
count = 0
while count > -1:
    print(count)
    count += 1
    if count > 5:
        break
```

Loops – Continue Statement

- Continue is used to skip the current block and go to the next iteration

```
# Prints out only odd numbers - 1,3,5,7,9
for x in range(10):
    # Check if x is even
    if x % 2 == 0:
        continue
    print(x)
```

Functions

- Defined using the keyword `def`, followed with the function's name
- Arguments are after the function name, in parathesis
- May return a value to the caller, using the keyword `return`

```
def my_function():
    print("Hello From My Function!")

def my_function_with_arguments(username, greeting):
    print("Hello, %s , From My Function!, I wish you %s"%(username, greeting))

def sum_two_numbers(a, b):
    return a + b
```

Functions

- Call a function by writing the function name with arguments

```
def my_function():
    print("Hello From My Function!")

def my_function_with_arguments(username, greeting):
    print("Hello, %s, From My Function!, I wish you %s"%(username, greeting))

def sum_two_numbers(a, b):
    return a + b

my_function()
my_function_with_args("John", “Merry Christmas!")
x = sum_two_numbers(1,2)  # x has the value 3
```

Functions – Default values

- Function arguments can have to have default values
- If the function is called without the argument, the argument gets its default value

```
def student(firstname, lastname ='Smith', grade ='fifth'):
     print(firstname, lastname, 'studies in', grade, 'grade')

student("Sam")
student("Sam", "Jones")
student("Sam", "Jones", "second")
```

Wake up!

- https://youtu.be/K72dRv5ejIc

Input from console

- The input function takes in a prompt and waits for console input
- The input entered at the console gets returned by input function
- Always returns a String type, must cast if other type is desired

```
name = input("What is your name? ")
print("Hello " + name)
print(type(name))
```

File Input/Output - Open

- open function opens a file and creates a file object
- file_object = open(file_name, access_mode, buffering)
  - file_name = name of file with filepath
  - access_mode (string) examples
    - r = read only
    - r+ = read and write
    - rb+ = read and write in binary format
    - w = write only
    - a = append
  - Buffering (not often used)

File Input/Output – File Object

- The *file* Object Attributes
  - file.closed - returns true if file is closed, false otherwise
  - file.mode - returns access mode with which file was opened
  - file.name – returns the file name

```
file = open("textfile.txt", "r+")
print "Name of the file: ", file.name
print "Closed or not : ", file.closed
print "Opening mode : ", file.mode
```

File Input/Output - Close

- `close` closes the file object and no more writing can be done
- flushes any unwritten information

```
# Open a file
file = open("file.txt", "r")
print "Name of the file: ", file.name

# Close opened file
file.close()
```

File Input/Output –With Open

- The `with` keyword is a more elegant way to open a file
- No need to call close

```
with open('file.txt') as file:
    print(file.name)
```

File Input/Output - Write

- The *write()* method writes any string to an open file
- Does not add a newline character ('\n') to the end of the string
- Note that Python strings can have binary data and not just text

```
with open("file.txt", "w") as file:
    file.write("Python is a great language.\nYeah its great!!\n")
```

File Input/Output - Read

- The *read()* method reads a file given the bytes to be read
- fileObject.read([count])
- If count is missing, it tries to read as much of the file as possible

```
with open("file.txt", "r") as file:
    file_text = file.read()
    print(file_text)
```

File Input/Output - Readline

- The *readline()* reads a single line at a time
- Requires a loop and counter

```
with open("file.txt", "r") as file:
    line = file.readline()
    count = 1
    file_contents = ""
    while line:
        file_contents += line
        line = file.readline()
        count += 1
    print(file_contents)
```

Next Time

- Classes and Objects
- Dictionaries
- Modules and Packages