OMIS 30: Intro to Programming (with Python)

Week 3, Class 1

Introduction to Programming
Instructor: Denis Vrdoljak

Learn programming with Python

Note Taker Needed!

If you take good notes, and would be willing to share your notes in exchange for a $60 giftcard or a letter of commendation, please sign up here:

https://www.scu.edu/disabilities/

Office Hours

Instructor	Days available
Yuan Wang (our TA)	M 2:30-3:30p, W 9-10a
Mike Davis (other section's instructor)	Tu 9:30-10:20a, Th 12-1p
Denis Vrdoljak	Tu 3:40-4:40p, W 5-6p

Course Topics

- ~~Computer setup~~
- ~~Shell - ls, mv (and rename), cp, pwd,cd, '..',mkdir, rm, touch, echo~~
- ~~cat, pipe, output redirect (> and >>), 'python --version' (introduce args)~~
- ~~Python Basics - print, input, math.~~
- Pseudo-code, algorithm design, comments
- iterables (~~lists~~, sets, dicts, ~~strings~~)
- Loops, Nested Loops, Recursion
- Flow Control (~~If, else, elif~~, try, except)
- Functions
- ~~Strings, upper(), lower()~~
- ~~indexing and slicing iterables~~
- lists, extending, ~~appending~~
- ~~mutability~~
- Jupyter Notebooks

Goals from last week

- Understand what dynamic typing means and how it works
- Understand Primitives and Iterables in Python
- Be able to index and slice strings and lists, and work with dictionaries
- Know some common methods for strings, lists and dictionaries
- Know how to print to standard out
- Know how to take in a user input
- Control a Program's flow with IF and WHILE loops

By the end of this week you should:

- Understand and be able to use While and For loops
- Know how to write a standalone Python script, and how to run it from the Command Line
- Have made progress on Project 1!

IF statement practice:

- What's the expected output?

```
x = [2,4,6,8]

if x:
    print(x)
else:
    print("No x to be found!")
```

IF statement practice:

- What's the expected output?

```
x = [2,4,6,8]

if x:
    print(x)
else:
    print("No x to be found!")
```

Output: [2,4,6,8]

IF, ELIF, ELSE example

OPEN FILES
- test_3.py
- test_2.py
- test.py

test_3.py | test_2.py

```
1  #get user input for favorite number
2  fav = input('What is your favorite number: ')
3
4  #print different results based on value of number
5  if int(fav) <= 50:
6      print("That's a good number")
7  elif int(fav) < 1000:
8      print("Interesting choice")
9  else:
10     print("That's a high number!")
11
```

```
Bens-MacBook-Pro:~ benthompson$ python ~/Desktop/test_3.py
What is your favorite number: 50
That's a good number
Bens-MacBook-Pro:~ benthompson$ python ~/Desktop/test_3.py
What is your favorite number: 500
Interesting choice
Bens-MacBook-Pro:~ benthompson$ python ~/Desktop/test_3.py
What is your favorite number: 1001
That's a high number!
Bens-MacBook-Pro:~ benthompson$
```

IF Statements REVIEW

- Creates a condition, which if True, executes the block of code. If False, it skips over it.
- Format:

```
if boolean_condition:
    #code to execute
```

- The block of code is indented with a tab (or 4 spaces)
- The IF block ends when the indentation ends

IF Statements REVIEW

- Creates a condition, which if True, executes the block of code. If False, it skips over it.

```
if my_animal == “mammal”:
    print(“It’s a mammal”)
```

- The block of code is indented with a tab (or 4 spaces)
- The IF block ends when the indentation ends

Nested IF Statements

- IF statements can be nested

```
If my_animal == “mammal”:
    if my_animal_species == “dog:
            print(“It’s a dog!”)
print(“It’s a mammal”)
```

IF-ELSE

- ELSE blocks execute if the IF condition is False

```
if my_animal == “mammal”:
    if my_animal_sepcies == “dog:
            print(“It’s a dog!”)
    else:
            print(“It’s a mammal, but not a dog”)
```

ELIF (else if)

- ELIF executes a second IF condition, if the first condition is False

```
if my_animal == “mammal”:
    if my_animal_sepcies == “dog:
            print(“It’s a dog!”)
    elif my_animal_sepcies == “cat”:
            print(“It’s a cat!”)
    else:
            print(“It’s a mammal, but not a dog or a cat”)
```

Equality Operators

equality ==

inequality !=

less than <

greater than >

less than or equal <=

greater than or equal >=

membership **in**

False vs. True

In python, everything evaluates as True by default unless it is empty or does not exist.

All empty sets, tuples, dictionaries, “0” evaluate to False

```
y = [] # an empty list
if y:
    print("y is full")
else:
    print("y is empty")
```

False vs. True

In python, everything evaluates as True by default unless it is empty or does not exist.

All empty sets, tuples, dictionaries, “0” evaluate to False

```
y = [] # an empty list
if y:
    print("y is full")
else:
    print("y is empty")
```

Output: y is empty

“is” vs “==”

- “==” tests for the same value, “is” tests for the same object

  Example:

```
x=5.0
y=5
print(x == y)
print(x is y)
```

“is” vs “==”

- “==” tests for the same value, “is” tests for the same object

  Example:

```
x=5.0
y=5
print(x == y)
print(x is y)
```

Output:
```
True
False
```

To check the exact ID of an object in memory, use print(id(x))

Boolean Operators

In Python, boolean operators (**and, or, not**) have lower **precedence** than the code chunks that they are comparing:

```
x = 11
y = 5
```

`x >= 11 and y < 6`	True, equivalent to (x >= 11) and (y < 6)
`x > 12 and y < 6`	False
`x > 12 or y < 6`	True
`x > 10 and not y < 3`	True

Pseudocode and Project 1

- Pseudocode = overall execution plan of the program
  - Can write a step by step guide on what the code needs to do
  - Can be a mixture of code and text
  - Meant to be human readable

WHILE loops

- Creates a condition and while it is True the block of code below is repeatedly executed. Once it is not true, the execution stops. (Checked in between loops.)
- Format:

  while boolean_condition:

      #code to execute

- The block of code is indented with a tab (or 4 spaces)
- The while block ends when the indentation ends

The format is the same a simple IF statement. It uses the same kinds of boolean conditions. But, a WHILE loop repeats until the boolean condition evaluates as False when it's next evaluated, while IF statements just execute once (if the boolean condition is true).

WHILE loops

Example:

```
>>    Index = 0
      while Index <10:
          #code to do
          print(Index)
          #increment index
          Index += 1
```

Output: (0,1,2,3,4,5,6,7,8,9)

WHILE loop teaching example

What if we want to check if every element in "x" is even?

```
x = [4,5,6,6.5,7]
i = 0
# i is commonly used for the index of an object
# while loops require explicitly instantiating the counter

while i < len(x):
    if x[i] % 2 == 0:
                        # % is called the modulus operator
                        # % yields the remainder from the division of the first argument by the second
        print(x[i], "is even")
    else:
        print(x[i], "is not even")

    i += 1
```

WHILE loop teaching example

```
x = [4,5,6,6.5,7]
i = 0

while i < len(x):
    if x[i] % 2 == 0:
        print(x[i], "is even")
    else:
        print(x[i], "is not even")

    i += 1
```

Output:

4 is even

5 is not even

6 is even

6.5 is not even

7 is not even

WHILE loops - Break

```
1  x = [0,1,2,3,4,5]
2  while True:
3      x.pop()
4      print(x)
```

```
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 2]
[0, 1]
[0]
[]

---------------------------------------------------------------------------
IndexError                                Trace
<ipython-input-70-bbc18fe3391b> in <module>()
      1 x = [0,1,2,3,4,5]
      2 while True:
----> 3     x.pop()
      4     print(x)

IndexError: pop from empty list
```

Without a stopping condition, the loop throws an error. Use **break** to gracefully stop a loop.

```
1  x = [0,1,2,3,4,5]
2  while True:
3      x.pop()
4      print(x)
5      if len(x) < 2:
6          break
```

```
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 2]
[0, 1]
[0]
```

WHILE loops - Continue

Using “continue” passes the control back to the top of the loop without exiting the loop, allowing certain conditions to not engage the rest of the loop.

```
1  x = [0,1,2,3,4,5]
2  while True:
3      x.pop()
4      if x[-1] == 2 or x[-1] == 4:
5          continue
6      print(x)
7      if len(x) < 2:
8          break
```

```
[0, 1, 2, 3]
[0, 1]
[0]
```

Nested WHILE loops:

Can insert while loops inside each other!

As more loops are ‘nested’ inside of each other, need to be more cognisant of accidentally making an infinite loop (a loop that never reaches the conditions to terminate).

Nested WHILE example (find the prime numbers):

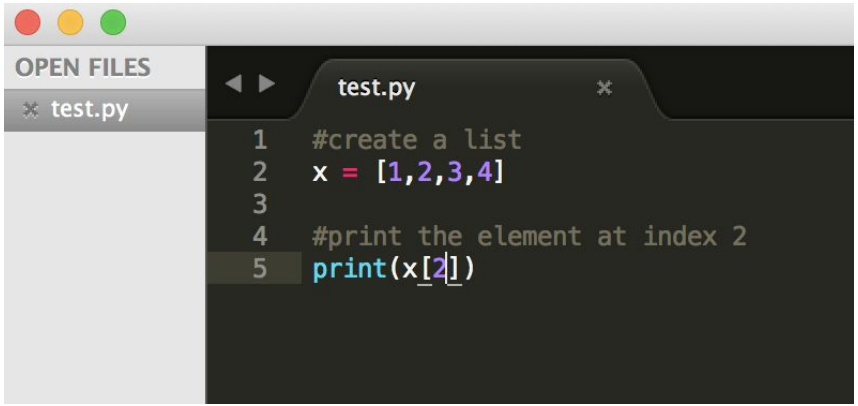
```
i = 2
while i < 100:
    j = 2
    while j <= i/j:
        if i%j == 0:
            break
        j += 1
    if j > i/j:
        print(i, "is prime")
    i += 1

print("Complete!")
```

Read this carefully, and try to figure out what's going on here. We didn't go through this one in detail in class, but going through this one is good practice to better understand while loops (even if you get a little lost in the algorithm for identifying primes).

Run Python from Command Line

- Create a .py file of the code that want to run
- Syntax to run it:

  >> python /path/to/file.py

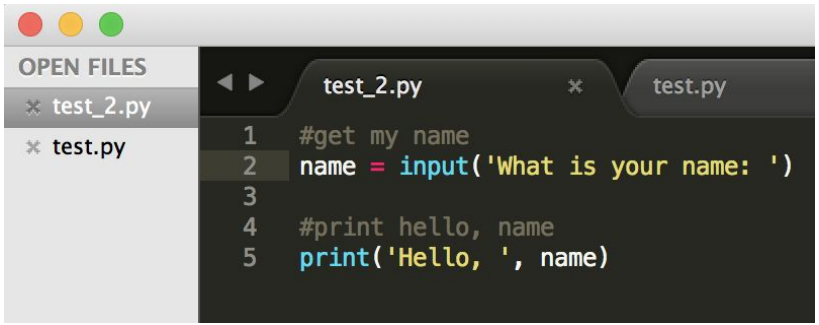



```
Last login: Mon Oct  1 10:43:25 on ttys003
You have new mail.
Bens-MacBook-Pro:~ benthompson$ python ~/Desktop/test.py
3
Bens-MacBook-Pro:~ benthompson$
```

Run Python from Command Line

- Create .py of the code that want to run
- Syntax to run it:

  >> python /path/to/file.py




```
Bens-MacBook-Pro:~ benthompson$ python ~/Desktop/test_2.py
What is your name: Ben
Hello,  Ben
Bens-MacBook-Pro:~ benthompson$
```

Homework 3.1

Submit via Camino

Full instructions:

- Create a standalone python script (.py file, not a .ipynb notebook) that asks a user for an input, does something (eg, prints user's input) repeatedly in an infinite loop, but quits if the first letter of the user input is a 'q' or a 'Q'
- Submit your .py file via Camino. Include your name as part of the filename.
- Submit the command to run your file as a separate .txt (text) file.

Appendix

- Reserved keywords in Python:
  - https://pentangle.net/python/handbook/node52.html
  - DO NOT USE THESE as VARIABLE NAMES!
- Dos/Windows vs Unix/Linux:
  - https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Step_by_Step_Guide/ap-doslinux.html
  - You should be familiar with the basic commands for navigating aroudn the file structure and modifying/creating files/folders. You should be aware of the harder to remember ones (like grep and vi) so you know what to Google when you need them!