Programming for Non-Programmers
Using Python

**Agenda**

6-7pm Python (Part 1)
Pizza and download data
7-8pm Python (Part 2)
Evaluation Form
Class Photo (Optional)

Online Links: [https://www.eecs.tufts.edu/~mshah08/lol.html](https://www.eecs.tufts.edu/~mshah08/lol.html)
Welcome to your introduction to computer programming! We are going to be learning a computer programming language, which you can think of as learning a new foreign language, just as if you were going to learn a new speaking language (Bonjour!). Just as there are hundreds of actively spoken languages in the world, there are hundreds of widely used programming languages used actively today.

The best way to learn is to dive in!

1. Course Description
We are going to be using the Python Programming Language. In order to get started programming, we will uncover the fundamental programming concepts that are found across almost every programming language. Once you master them in Python, you can fully utilize them in Python or any other language you may choose to learn.

Note: On Learning to Program
I encourage you to go ahead and type in the examples provided. Then spend time playing and modifying them to your liking. If the program fails to do what you intended, simply restart, or at the very least copy & paste in the code and try to follow through the programs execution.

Let us begin!

2. Getting Setup
This might be your first time using Linux! Mac users may feel a bit more comfortable, but there is no need to fear, Linux was built by programmers for (soon to be) programmers!

Step 1. Open up a terminal (System->System Accessories->Terminal)
Step 2. Type in ‘idle&’ to open up the Python IDLE editor which we will be using.
3. Data Types

Arithmetic
As you imagine, the other way to represent data is with numbers. We can perform operations such as addition (`+`), subtraction (`-`), multiplication (`*`), and division (`/`).

Try typing these commands into the IDLE editor now and pressing enter.

1. 2+5
2. 7-5
3. 5*4+2
4. 8/2*4
5. 2.7 *3

In math, we know there are different numbers, such as whole numbers, real numbers, integers, complex numbers, etc. The two most common in Python are floats (numbers with decimals) and int’s (integers).

Let’s see a quick example and introduce the print command.

More than you needed to know right now: `print` is a function that outputs the contents following the command out to a console.

1. print(int(5))
2. print(float(5)) #Note, you will see the value 5.0, because we are representing a decimal number.

Numbers themselves represent one of the fundamental ways to represent data in Python, but we can also represent data as text.

Strings
The second way to represent data as a string, which is a piece of data that represents text. A string is individually made up of a collection of one or more characters (`A-Z`, `1-9`, `$`, `#`, etc.).

Try typing in these commands in IDLE

1. print(“hello world”)

^The string is the part that is represented in between the double quotes. And congratulations! You just wrote your first real program, the notoriously famous HELLO WORLD program!

Let’s try some more examples.
1. `print('abcdefg')`
   # Note we can use single or double quotes around a string, but we cannot mix them.

2. `print('123456')`
   # Note: That the data type of the items between the quotes is of a string. If we want it to be represented as a number, we have to tell python to try to cast (i.e. transform) it into another type.

3. `print(int('123456'))`
   # Same result as above, but this time an integer is returned.

Should I type in the pound (#) sign? Any text behind the # sign gets ignored by Python. This means you can write comments for yourself to remember what exactly you were trying to achieve. It’s a great habit to write comments in your code.

Strings are one of the fundamental ways to represent data.

**Variables**

A variable is a container for data. It is a way to refer to some piece of data. Here are some simple examples.

```python
that_person = “Mike”
```

Here we have a variable called that_person is assigned to the value Mike. We use the equals operator to assign what is on the left of the equation (that_person) to what is on the right of the equation (“Mike”).
Lists

A list is a versatile data structure in Python, in which we can store a sequence of data.

To create a list, we name it, just like we would a variable. We then list each element between brackets [ and ]. Each element in our list is then separated with a comma.

```
BestFriends = ['Willie','Mike','Tomoki']
```

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Willie</td>
</tr>
<tr>
<td>1</td>
<td>Mike</td>
</tr>
<tr>
<td>2</td>
<td>Tomoki</td>
</tr>
</tbody>
</table>

We can access elements individually by doing the following.

```
input > BestFriends[2]
Output > "Tomoki"
```

We can also add elements to our list, by appending them. When we append to a list, we update it by adding an element at the end.

```
BestFriends.append("Raoul")
```

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BestFriends[0]</td>
<td>Willie</td>
</tr>
<tr>
<td>BestFriends[1]</td>
<td>Mike</td>
</tr>
</tbody>
</table>

There are some other common list operations we may want to perform listed in this table.

<table>
<thead>
<tr>
<th>Python Code</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len(BestFriends)</code></td>
<td>Get length of the list</td>
<td>4</td>
</tr>
<tr>
<td><code>[1,2]+[3,4]</code></td>
<td>Concatenate two lists</td>
<td>[1,2,3,4]</td>
</tr>
<tr>
<td>'Mike' in BestFriends</td>
<td>Test for membership in list</td>
<td>True</td>
</tr>
<tr>
<td><code>for x in BestFriends:</code></td>
<td>Iterate through all of the elements of the list.</td>
<td>Willie Mike Tomoki Raoul</td>
</tr>
</tbody>
</table>
4. Program Constructs

Control Flow

Computers themselves are extremely obedient! From our previous examples, you can see how as soon as we enter a command, the computer will execute it right away without even thinking!

We may want to simulate some way within our programs to execute commands based on certain conditions. This is known as ‘control flow’, and we can visualize it as a graph.

We are going to introduce one more way to represent data, which is the Boolean. The Boolean stores a value of either true or false. Internally, a computer represents the value as a 1 (for true) or a 0 (for false).

We can use these values to then follow different paths in our control flow.

Conditional Statement

A condition statement is a statement that if a condition evaluates to true, then the block of code below it will execute.

Here’s a simple example combining Booleans and control flow.

joe_married = 0
if joe_married:
    print('Status: Married')
else:
    print('Status: Single')

Another example of control flow introducing ‘elif’, which says, “if the first condition is not true, then check if this is true, else execute the following code by default”.

joe_married = 2
if joe_married == 0:
    print('Status: Married')
elif joe_married == 1:
    print('Status: Single')
else:
    print("Status: It’s complicated")
#Note that we have to use double quotes in this example to wrap our string.

```python
>>> joe_married = 1
>>> if joe_married == 0:
    print('Status: Married')
elif joe_married == 1:
    print('Status: Single')
else:
    print("Status: It's complicated")

Status: Single
>>> joe_married = 3523532
>>> if joe_married == 0:
    print('Status: Married')
elif joe_married == 1:
    print('Status: Single')
else:
    print("Status: It's complicated")

Status: It's complicated
```

Instead of always using the ‘==’ operator which tests if two values are equal, we can also use the following operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>1 &lt; 2</td>
<td>True</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>2 &gt; 2</td>
<td>False</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>2 &lt;= 2</td>
<td>True</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>3 &gt;=2</td>
<td>True</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
<td>1==2</td>
<td>False</td>
</tr>
<tr>
<td>!= or &lt;&gt;</td>
<td>Not equal to</td>
<td>1 != 2</td>
<td>True</td>
</tr>
</tbody>
</table>

*More than you needed to know right now: Testing equality between an int and a string could be problematic. Testing equality between a float and a float could also be problematic! In all cases, it has to do with how computers represent data behind the scenes.*

*Quick Review: Representing Data in Python*

String – A collection of characters (More correctly stated: a list of characters)

Number – An int (..., -3, -2, -1, 0, 1, 2, 3, ...) or a float (Decimal value such as 3.14)

Boolean – 1 (A true value) or 0 (A false value). We can alternatively use the keywords ‘True’ or ‘False’
Indentation

You might have noticed me hitting the tab key. In Python, indentation is important! It makes the code more readable. When we write code, we segment it into separate blocks of code that will run.

Functions

We’re starting to type a lot of text, and a long time ago programmers realized it is important to reuse code for their own sanity. A function in Python is similar to one we are use to in mathematics, such as \( y(x) = x^*x \). We have a function ‘y’ that takes some parameter ‘x’, and then it returns a value based on the parameter x.

Let’s go ahead and write that function using Python.

```python
def square(x):
    return x*x
```

We define a new function with the Python languages ‘def’ keyword, give it a name ‘square’, and then any parameters that will be passed in. Finally we (optionally) return a value.

Let’s write our `is_joe_married` function as well.

```python
#Note we do not return anything in this function.
def is_joe_married(x):
    if x == 0:
        print('Status: Married')
    elif x ==1:
        print('Status: Single')
    else:
        print("Status: It's Complicated")
```
```python
>>> def is_joe Married(x):
    if x == 0:
        print('Status: Married')
    elif x == 1:
        print('Status: Single')
    else:
        print("Status: It's Complicated")

>>> is_joe Married(0)
Status: Married
>>> is_joe Married(1)
Status: Single
>>> is_joe Married(2)
Status: It's Complicated
>>> is_joe Married(-234)
Status: It's Complicated
>>>```
**Loops**
Loops are an important part of programming. They are a type control flow, in which we test a condition, and if it is true we continue to execute within the loop.

**While Loop**
A while loop is a piece of code the executes while some condition is true at the top of the loop. Here’s a few examples.

```
count = 0
while count < 10:
    print('count is: ' + str(count))
    count = count + 1
```

```
>>> count = 0
>>> while count < 10:
    print('count is: ' + str(count))
    count = count + 1

  count is: 0
  count is: 1
  count is: 2
  count is: 3
  count is: 4
  count is: 5
  count is: 6
  count is: 7
  count is: 8
  count is: 9
```

**For Loop**
A for loop is a piece of code that executes for a specified range. Semantically, we can achieve the same functionality as a while loop.

Iterating through a list:
```
for x in BestFriends:
    print(x)
```

Iterating through a list again over length of list:
```
for x in range(len(BestFriends)):
    print(x)
```

```
>>> for x in BestFriends:
    print(x)

  Willie
  Mike
  Tomoki
  Raoul
```

```
>>> for x in range(len(BestFriends)):
    print(x)

  0
  1
  2
  3
```

#Note the above seems unintuitive, but x is a variable that we’re iterating through. So x is incrementing by one each time we find an element in the loop.
Lets try that again with what we learned about accessing a list.

```python
for x in range(len(BestFriends)):
    print(BestFriends[x])
```

```
Willie
Mike
Tomcki
Raoul
```
5. Programming Challenge

Program 1: Guessing Game!
Goal:
- Randomly generate a number from 1 to 10.
- Take user input from user to guess the randomly generated number.
- When the user guesses the correct answer, output that they have finished.

Hint: To create a random number use the following Python function

```python
import random # Bring in Python functions
answer = random.randint(1,10) #Includes both 1 and 10
```

Hint: To get user input(stored as a string), use the ‘input’ command, but remember when using guess to cast it to an int if you do any comparisons.

```python
guess = input('What is your guess?
```

Program 1: Hints and suggestions
- You will need one loop that runs until the user guess matches the computer guess
- You will need two variables, one that keeps track of the answer and the guess.
- Use if statements to help the user make better guesses.
- Keep track of the number of guesses the user makes.
  - If they do it in less than 4 tell them they win!

Writing code in a window and then running
1.) Hit Ctrl+N or navigate to File and click ‘New Window'

2.) You will see a new blank window appear.
You can type code in here.
3.) When you are ready to run, press ‘F5’ or navigate to ‘Run Module’ and it will run in the previous *Python Shell* Window. You may be warned to save your program if you have not already done so.

The solution will be on the next page! Please do feel free to look after you’ve made at least one attempt.
import random

answer = random.randint(1,10)
guesses = 0
guess = -1

while guess != answer:
    guess = input('What is your guess?')
    if int(guess) < answer:
        print('Try a higher number')
    elif int(guess) > answer:
        print('Try a lower number')
    guesses = guesses + 1

print('You got it in '+str(guesses)+'!!')

if guesses<5:
    print('You are smart')
else:
    print('Luck was not on your side')
**Going Further**
Did you finish with extra time left, and want to do more before the next lesson?
- Try adding another loop and asking the user if they’d like to play again.
- Add another variable to keep track of the least number of guesses.
  - Perhaps increase the range to (1,100) to make this more interesting.
- Personalize the game, so that the first thing we do is ask for a name.
  - Add that name to a ‘BestGuessingFriends’ list.

6. **Appendix**

**Low Level Details (More than you need to know now)**
Python is an interpreted programming language. What this means, is that the computer reads Python code one line at a time and executes it. If you h

**File IO**
Some useful snippets on opening files (to be covered by Willie in lesson 2 anyway).

```python
with open('filepath') as f:
    lines = f.read().splitlines()
```