Intro to Python

Why Python?

```
print("Hello world!")
```

```
Hello world!
```

Python Environment

(Denis)

Programming in Python can look like many things. Often times, it is interacted with as a command line interface (sometimes also referred to as a "console", a "Python shell", a "REPL"). Other times, people use it in an iPython or Jupyter notebook (that's what this document is), which allows users to selectively execute written code.

There are also other more sophisticated python development environments, such as PyCharm

In general, a Python environment listens for user input, evaluates that input, and returns whatever the result is.

Try running the code blocks below (click on one and press Shift-Enter). The line that appears below the block is what is returned as a result of running that code.

```
3 + 4 ## Addition
```

```
7
```

```
5**2 ## An exponent operation
```

```
25
```

```
5/6 ## Decimal/"floating point" division
```

```
0.8333333333333334
```

```
5//6 ## Integer division (note the difference compared to above!)
```

```
0
```

```
not True ## Negating a boolean
```

```
False
```

```
not False
```

```
True
```

Our inputs to Python can roughly be categorized as either **commands** (i.e., requests to do something), or **definitions**

```
## Two commands
print(4 + 7)
```

```
11
```

```
## Defining a variable
my_location = "New York University"
```

```
## Combining them!
print(my_location)
```

```
New York University
```

Side note: Jupyter notebooks automatically print the last result:

```
4+5
2+3
```

```
5
```

Introduction to Data Types

(Nick)

Let's focus on **variable definitions** for a moment.

One of the most common things you will be doing is defining the data that you will be using in the course of programming

```
## Whenever we want to use data, we declare and assign it to a "variable", in this case, my_city
my_city = "New York"
```

```
current_floor = 6
```

Above, we've defined two variables. The "single equals" sign performs an assignment operation. One way to think of this is:

> "The variable 'current_floor' *gets* the value 6".

Notice that the actual data (right-side of the equals sign) is being highlighted in different colors across the two variables. This is happening because one variable contains text, or **string** data, while the other contains a number (an **integer**)

These are two of the essential "data types".

```
## For instance, say we are currently on floor 6 of this building, but
## we know that after this tutorial is finished, there's a book we want
## to check out, and it's on the ninth floor.

destination_floor = 9
```

```
## We can figure out how many floors we'll have to travel to get there

floors_I_need_to_walk = destination_floor - current_floor
```

```
print(floors_I_need_to_walk)
```

```
3
```

Using variables allows us to write generalized code that can serve as a formula, even as the specific values saved in these variables change.

List of data types

- string : `"Hello there"`
- boolean (true/false) : `True`
- integer : `4`
- decimal (float) : `4.0`

Different operations are defined for each data type. For instance, you can *negate* any boolean value (turn a `True` into a `False`, or vice versa); you can *concatenate* strings together; you can *add* or *subtract* integers and decimals.

You can check the type of any data using the `type()` function

```
print( type(4) )
print( type("Hello") )
```

```
<class 'int'>
<class 'str'>
```

Reserved Keywords

In Python, some words are "reserved" and cannot be used as names for variables or functions (identifiers).

For instance, the word `True`, when it does not appear within quotation marks, will always evaluate to the *value* **True** (and similar for `False`). Other reserved tokens are those that comprise the actual language, such as `if`, `else`, `return`, and so on.

https://www.programiz.com/python-programming/keyword-list

```
True
```

```
True
```

```
var1 = 10
var2 = "Hey!"
True = 3.4 ## Doesn't work!
```

```
  File "<ipython-input-19-8c226123a853>", line 3
    True = 3.4 ## Doesn't work!
    ^
SyntaxError: cannot assign to True
```

SEARCH STACK OVERFLOW

```
var3 = True ## This, however, is completely fine...
var3
```

```
True
```

Conditional Statements

(Denis)

Conditional if-statements are a very big part of programing with any language. We can think about it first using some pseudocode:

```
if ( it is raining outside ):
  grab an umbrella before leaving
else:
  leave the umbrella at home
```

Conditional statements are a way to specify sections of code that should be executed only when a condition is true. You can think of the if and else lines as being gatekeepers to the code that exists right below them.

```
## Only when the condition between the parentheses evaluates to True do we
## evaluate the associated code in the body

iHaveMoney = False;

if iHaveMoney:
    print("I will buy ice cream!")
else:
    print("I can't buy ice cream.")
```

```
I can't buy ice cream.
```

```
friendHasMoney = False

if iHaveMoney or friendHasMoney:
    print("We will buy ice cream!")
else:
    print("We can't buy ice cream.")
```

```
We can't buy ice cream.
```

Multiple conditional statements can be written between the paranthesis, joined by the words "or" and "and". These words are logical operators in Python.

As long as the statement within the paranthesis evaluates to a logical value (i.e., either **True** or **False**), it can be used within an if-statement.

```
3 > 5 and 3 > 1
```

```
False
```

```
3 > 5 or 3 > 1
```

```
True
```

Statements on *both* sides of an "and" must evaluate to `True` for the interpreter to enter the body of code following it.

Challenge 1

```
if True or False:
  print("A")

if True and False:
  print("B")
```

What will be printed as a result of running the above code block?

1. A

   B
2. A
3. B
4.

```
## Try it out!

if True or False:
    print("A")

if True and False:
    print("B")
```

```
A
```

```
## Testing equality

print( 3 == (4 - 1) )
## use the double equals to test equality, since single equals
## is only used for assigning values

print (3 != (4 - 1) )
## the != operator means 'not equal'
```

```
True
False
```

```
hour = 10
if hour > 7 and hour <= 12:
    print("Good morning!")
elif hour > 12 and hour <= 18:
    print("Good afternoon!")
elif hour > 18 and hour <= 22:
    print("Good evening!")
else:
    print("You're too late or too early!")
```

```
Good morning!
```

Nesting if-statements

If-statements can be arbitrarily nested in your code:

```
## For this example, we'll import a function that generates random values
from random import *
```

```
## Here we use the random number generator to find some value
## between 0 and 1
random_number = random()
print("My random number is: " + str(random_number))
```

```
My random number is: 0.6159633901610608
```

```
if random_number < 0.5:
    print("The number is less than .5")
    if random_number < 0.25:
        print("... in fact, it's even less than .25")
else:
    print("The number is greater than (or equal to) .5")
```

```
The number is greater than (or equal to) .5
```

Click here for an illustration of the logic in the if-statement above.

What is the difference between Else-If and Nested If statements?

Essential Data Structures

(Denis)

Lists

```
## Let's say I'm teaching a tutorial, and I want to save the names of all of my students

## One way is to create a variable for each one, and save one name per variable

student_1 = "Alice"
student_2 = "Xiaojing"
student_3 = "Frank"

## This works ok... but it makes more sense to keep a single list!

my_students = ["Alice", "Xiaojing", "Frank", "George", "Amy", "Anita"]
```

```
my_students
```

```
['Alice', 'Xiaojing', 'Frank', 'George', 'Amy', 'Anita']
```

Elements within a list can be accessed by their *index number*

Index numbers start at index 0, meaning that the first element in the list is at element 0

```
my_students[0] ## You access elements of a list using this notation
my_students[1]

## You can even use an expression in the place of the index, as long
## as that expression turns into a valid index
my_index = 1 + 2
print(my_students[my_index])
```

```
George
```

```
my_students[0:3] ## Take a subset of the list, starting at 0, stopping before 3
```

```
['Alice', 'Xiaojing', 'Frank']
```

```
my_students[-1] ## Access the list from the right (the end of it) using a negative index
```

```
'Anita'
```

```
my_students[::2] ## Step through 2 at a time
```

```
['Alice', 'Frank', 'Amy']
```

```
my_students[::-1] ## Reverse the list
```

```
['Anita', 'Amy', 'George', 'Frank', 'Xiaojing', 'Alice']
```

Note: we reversed the list above, but what happens when you look at the value of `my_students` again?

```
my_students
```

```
['Alice', 'Xiaojing', 'Frank', 'George', 'Amy', 'Anita']
```

```
my_students.append("Jacob") ## Add a value to the end of a list

my_students
```

```
['Alice', 'Xiaojing', 'Frank', 'George', 'Amy', 'Anita', 'Jacob']
```

```
my_students.pop(2) # remove element by index
my_students
```

```
['Alice', 'Xiaojing', 'George', 'Amy', 'Anita', 'Jacob']
```

```
late_registrations = [ 'Sally', 'Jose' ]

## Here we are concatenating two lists
## The `+` sign means addition for integers, but on lists it means concatenate

print(my_students + late_registrations)
print(my_students)
```

```
['Alice', 'Xiaojing', 'George', 'Amy', 'Anita', 'Jacob', 'Sally', 'Jose']
['Alice', 'Xiaojing', 'George', 'Amy', 'Anita', 'Jacob']
```

```
my_list = my_students.copy()
print(my_list)
my_list.append(late_registrations)
print(my_list)
print(my_students+late_registrations)
```

```
['Alice', 'Xiaojing', 'George', 'Amy', 'Anita', 'Jacob']
['Alice', 'Xiaojing', 'George', 'Amy', 'Anita', 'Jacob', ['Sally', 'Jose']]
['Alice', 'Xiaojing', 'George', 'Amy', 'Anita', 'Jacob', 'Sally', 'Jose']
```

```
## We are saving the result back to the variable called `my_students`
my_students = my_students + late_registrations
```

```
## Checking the length of a list
len(my_students)
```

```
8
```

Testing membership of a list

```
if 'Anita' in my_students:
    print("What's up Anita!")
else:
    print("Oh, is Anita not in this class?")
```

```
 What's up Anita!
```

Challenge 2

Write some code that picks a student from your list `my_students` at random

Hint: you can always round down a decimal number to an integer with the `int(x)` function:

```
int( 4.6 ) ## this equals 4
```

```
my_students
```

```
['Alice', 'Xiaojing', 'George', 'Amy', 'Anita', 'Jacob', 'Sally', 'Jose']
```

Dictionaries

(Nick)

While lists are a great way to store any sort of ordered data that it makes sense to access via a numerical index, sometimes we want to find pieces of data based on other types of input, such as a string.

Another way to put this is that, given some **key**, we want to find the coresponding **value**.

For example, it might be easier to keep my dataset of students in a form where I can find someone's name given their NYU net id.

```
netid_students = {
    "az123": "Alice",
    "xa746": "Xiaojing",
    "fj32" : "Frank",
    "gm330": "George",
    "aa21" : "Amy",
    "ar987": "Anita",
    "g3745": "George"
}

## Now we can retrieve a name by using a netID as a key
print(netid_students['aa21'])
```

```
Amy
```

We can actually store anything we want as values in a dictionary, including lists, strings, numbers, or even other dictionaries.

Using dictionaries as records

Often, people use dictionaries to store different types of information about whatever they are trying to model:

```
student1 = {
    "name" : "Alice",
    "netid": "az123",
    "currently_enrolled": True,
    "credits_completed": 32,
    "school": "GSAS",
    "enrolled_in": ["Intro to Python", "Fundamental Algorithms"]
}

student2 = {
    "name" : "Xiaojing",
    "netid": "xa746",
    "currently_enrolled": True,
    "credits_completed": 16,
    "school": "Wagner",
    "enrolled_in": [
        "Intro to Python",
        "Natural Language Processing",
        "Urban Planning",
        "Shakespeare"]
}

student_profiles = [student1, student2]
```

Loops

(Nick)

Looping is one of the most fundamental operations in programming, but can be tricky to learn for the first time. Python has several ways of performing loops.

```
print(my_students)
```

```
['Alice', 'Xiaojing', 'George', 'Amy', 'Anita', 'Jacob', 'Sally', 'Jose']
```

```
## Print each element of a list
for student_name in my_students:
    print(student_name)
```

```
Alice
Xiaojing
George
Amy
Anita
Jacob
Sally
Jose
```

What's happening in the above example? The line `for student_name in my_students:` defines a **loop** through every element in `my_students`.

Each element gets assigned to the temporary variable `student_name`, and then the body of the loop is executed; in this case, the body is only printing the value of that temporary name.

Also notice that the line `print(student_name)` is indented. This is an aspect of Python –– the body of any loop, conditional, or function definition always needs to be indented.

```
## We use a function to automatically create a list of numbers, starting at 0,
## and ending at 19
my_numbers = list(range(0,20))

my_numbers
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
for number in my_numbers:
    if number == 5:
        print("Hey, I love 5!")
    else:
        print(number)
```

```
0
1
2
3
4
Hey, I love 5!
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

Not only can we loop through lists, we can loop through dictionaries. With lists, we loop through them one element at a time (as seen above). Looping through dictionaries is slightly different –– we iterate through it one **key-value** pair at a time:

```
## Notice that in order to loop through a dictionary's key-value pairs, we have
## to append .items() to the name of the dictionary

for netid, name in netid_students.items():
    print("The student with netid: " + netid + " has the first name " + name)
```

```
The student with netid: az123 has the first name Alice
The student with netid: xa746 has the first name Xiaojing
The student with netid: fj32 has the first name Frank
The student with netid: gm330 has the first name George
The student with netid: aa21 has the first name Amy
```

```
The student with netid: ar987 has the first name Anita
The student with netid: g3745 has the first name George
```

```
print(student_profiles)
```

```
[{'name': 'Alice', 'netid': 'az123', 'currently_enrolled': True, 'credits_completed': 32, 'school': 'GSAS', 'enrolled_in': ['Intro to P
```

```
## Looping through a list that contains dictonaries

for student in student_profiles:
    if "Fundamental Algorithms" in student['enrolled_in']:
        print(student['name'])
```

```
Alice
```

We can also loop indefinitely, as long as some constraint remains true, using a **while-loop**:

```
x = 0
while x < 100:
    print(x)
    x = x + 10
    ## Don't forget to update the value of x or you will be stuck in
    ## an infinite loop!
```

```
0
10
20
30
40
50
60
70
80
90
```

In general, while-loops are helpful for performing some activity when you don't know before you start how many times the loop will need to run. For-loops are helpful in cases when you do know exactly how many times you want it to run.

Text strings are simply lists of characters, so you can loop through them as well:

What does this do?

```
word = 'lead'
for char in word:
    print(char)

(a) char
(b) lead
(c)     lead
(d) aaaa
(e) rrrr
(f) l
    e
    a
    d
```

Working With Strings

(Denis)

Python comes with many pre-defined functions that work on text strings. Combined with the ability to traverse through a string using loops (as shown above), there are tons ways to work with text.

```
```

```
## Convert text to all lowercase or uppercase

my_string = "Hello World!"

print(my_string)
print(my_string.lower())
print(my_string.upper())
```

```
Hello World!
hello world!
HELLO WORLD!
```

```
## Split a string

intro_text = ("Python is a widely used high-level programming language for general-purpose programming, "
              "created by Guido van Rossum and first released in 1991.")

tokens = intro_text.split(" ") ## Here, we split on the whitespace character
tokens ## We end up with a list of tokens
```

```
['Python',
 'is',
 'a',
 'widely',
 'used',
 'high-level',
 'programming',
 'language',
 'for',
 'general-purpose',
 'programming,',
 'created',
 'by',
 'Guido',
 'van',
 'Rossum',
 'and',
 'first',
 'released',
 'in',
 '1991.']
```

```
## Joining a list into a string
## (the reverse of a split!)

print(" ".join(tokens))

print(", ".join(my_students))
```

```
Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released
Alice, Xiaojing, George, Amy, Anita, Jacob, Sally, Jose
```

```
## Making substitutions with replace()
addresses = "124 Fake St., 325 Broadway Ave., 718 Washington Ave., 70 W. 4th St."
print(addresses)

normalized_addresses = addresses.replace("St.","Street").replace("Ave.","Avenue")
print(normalized_addresses)
```

```
124 Fake St., 325 Broadway Ave., 718 Washington Ave., 70 W. 4th St.
124 Fake Street, 325 Broadway Avenue, 718 Washington Avenue, 70 W. 4th Street
```

```
## Reversing a string
print(normalized_addresses[::-1])
## This uses the slice notation, with start and end set to the beginning and end
## of the string, but the "step" parameter set to -1
```

```
teertS ht4 .W 07 ,eunevA notgnihsaW 817 ,eunevA yawdaorB 523 ,teertS ekaF 421
```

But what if our strings have quotes in them? How do we print a statement like this:

> Wait, she's going to say "Hello!"

```
print("Wait, she's going to say \"Hello!\"")
print('Wait, she\'s going to say "Hello!"')
print("""Wait, she's going to say "Hello!" """)
```

```
Wait, she's going to say "Hello!"
Wait, she's going to say "Hello!"
Wait, she's going to say "Hello!"
```

Strings can be created in a number of different ways:

```
'single quotes'
"double quotes"
""" triple-double quotes """  #This can contain line breaks!
```

or even like this:

```
''' triple-single
    quotes '''
```

Functions

(Nick)

A user may want to define a function for a sequence of calculations needed to be executed at multiple different places in code. Functions make programming more concise and can serve as a great tool for the user writing code.

Functions are defined with the following syntax:

```
def f (x):
    print(x)
```

```
def   tells Python that you are writing a function definition. This line of code is also followed by a colon
f     is the name of the function so that the user can call the function later on
x     within the parenthesis are parameters passed into the function to be used
```

The proceeding lines of code are the body of a function and should be indented

```
##Suppose we have the following String variables
x = "You just called a function on x"
y = "You just called a function on y"
```

```
##We can call our function f on each x and y
f(y)
f(x)
```

```
You just called a function on y
You just called a function on x
```

Notice how function f simply prints the parameter we pass in. Alternatively, functions can have a return value, which is something that may be retrieved or calculated within the function

```
##Suppose we have a function that adds 5 to a given integer
def addFive(x):
    x = x + 5  #In this line, we are assigning a new value to x
    return x   #Return is what the function will give back
```

```
##Now let's call addFive on an integer 5
addFive(50)
```

```
55
```

```
##What does this return?
5 + addFive(5)
```

```
15
```

```
##What does this return?
addFive(addFive(5))
```

```
15
```

Functions can be nested, and are evaluated from inside to out when implemented so

```
##Let's say I want a function that adds 5 to only negative integers
def add_to_negatives(x):
    if (x < 0):
       return x + 5
    else:
        return x
```

```
##What is the output of calling
add_to_negatives(-2) - add_to_negatives(3)
```

```
0
```

There is no rule to how complex a function definition should be, but the key is to have it concise so that it could be called on often

```
##Functions can be as simple as a check to see if a number has more than two digits
def digits_check(x):
    if (x > 99 or x < -99):
        return True
    else:
        return False
```

```
digits_check(100)
```

```
True
```

```
digits_check(-88)
```

```
False
```

```
# functions are re-usable pieces of code that can make your script more compact
# functions also make it easier to compartmentalize the code
def welcome_time(hour):
    if hour > 7 and hour <= 12:
        return('Good morning!')
    elif hour > 12 and hour <= 18:
        return('Good afternoon!')
    elif hour > 18 and hour <= 22:
        return('Good evening!')
    else:
        return('You\'re too late or too early!')

print(welcome_time(10))
```

```
Good morning!
```

```
def welcome(name,h):
  print(welcome_time(h).replace('!',', ' + name + '!'))

welcome('Paul',10)
welcome('Sally',20)
welcome('Gene',5)
```

```
Good morning, Paul!
Good evening, Sally!
You're too late or too early, Gene!
```

(Group) Challenge 3

Background: You're doing a project on linguistics and want a way to find all of the palindromes that commonly appear in pieces of literature. But where to begin???

Part 1

Write a function, `is_palindrome(x)` that takes a single word (as a string) as input, and returns either `True` or `False` depending on if the input is a palindrome

e.g. `is_palindrome("Hello world!")` should return **`False`**.

however, `is_palindrome("tacocat")` should return **`True`**.

```
def is_palindrome(input):
    ## Fill this in!
    return False
```

Part 2

Using `is_palindrome(x)`, write a new function that takes a paragraph of text as input, checks each individual word larger than 1 character to see if it is a palindrome, and returns a list of all of the palindromes which were found.

e.g.

`palindrome_search("I have no idea what a tacocat is.") = ['tacocat']`

```
## Two paragraphs from the first pages of Moby Dick:
para1 = ("Call me Ishmael. Some years ago—never mind how long precisely—having "
         "little or no money in my purse, and nothing particular to interest me "
         "on shore, I thought I would sail about a little and see the watery part "
         "of the world. It is a way I have of driving off the spleen and regulating "
         "the circulation. Whenever I find myself growing grim about the mouth; whenever "
         "it is a damp, drizzly November in my soul; whenever I find myself involuntarily "
         "pausing before coffin warehouses, and bringing up the rear of every funeral I meet; "
         "and especially whenever my hypos get such an upper hand of me, that it requires a strong "
         "moral principle to prevent me from deliberately stepping into the street, and methodically "
         "knocking people’s hats off—then, I account it high time to get to sea as soon as I can. This "
         "is my substitute for pistol and ball. With a philosophical flourish Cato throws himself upon his "
         "sword; I quietly take to the ship. There is nothing surprising in this. If they but knew it, almost "
         "all men in their degree, some time or other, cherish very nearly the same feelings towards the ocean with me.")
para2 = ("But here is an artist. He desires to paint you the dreamiest, shadiest, "
         "quietest, most enchanting bit of romantic landscape in all the valley of the "
         "Saco. What is the chief element he employs? There stand his trees, each with a "
         "hollow trunk, as if a hermit and a crucifix were within; and here sleeps his meadow, "
         "and there sleep his cattle; and up from yonder cottage goes a sleepy smoke. Deep into "
         "distant woodlands winds a mazy way, reaching to overlapping spurs of mountains bathed in "
         "their hill-side blue. But though the picture lies thus tranced, and though this pine-tree "
         "shakes down its sighs like leaves upon this shepherd’s head, yet all were vain, unless the "
         "shepherd’s eye were fixed upon the magic stream before him. Go visit the Prairies in June, when "
         "for scores on scores of miles you wade knee-deep among Tiger-lilies—what is the one charm wanting?—Water—there "
         "is not a drop of water there! Were Niagara but a cataract of sand, would you travel your thousand miles to see it? "
         "Why did the poor poet of Tennessee, upon suddenly receiving two handfuls of silver, deliberate whether to buy him a coat, "
         "which he sadly needed, or invest his money in a pedestrian trip to Rockaway Beach? Why is almost every robust healthy boy with "
         "a robust healthy soul in him, at some time or other crazy to go to sea? Why upon your first voyage as a passenger, did you yourself
         "feel such a mystical vibration, when first told that you and your ship were now out of sight of land? Why did the old Persians hold
         "the sea holy? Why did the Greeks give it a separate deity, and own brother of Jove? Surely all this is not without meaning. And sti
         "deeper the meaning of that story of Narcissus, who because he could not grasp the tormenting, mild image he saw in the fountain, pl
         "into it and was drowned. But that same image, we ourselves see in all rivers and oceans. It is the image of the ungraspable phantom
         "and this is the key to it all.")


def palindrome_search(paragraph):
    ## Fill this in!
    return []
```

Challenge 4

Write a function, `vowel_count(x)` that takes a string as input, and outputs a count of the number of vowels in that string

e.g. `vowel_count("Hello world!")` should return `3`.

```
def vowel_count(x):
  ## Do something!
  return 0
```

(Extra) Challenge 5

Write a function, `random_value(list)` that takes a list of anything, and returns a random element from it

Hint: this is very similar to the challenge before, about picking a random student, but this time it needs to work for all possible lists!

e.g.

```
northeast = ["New York",
             "New Jersey",
             "Pennsylvania",
             "Connecticut",
             "Vermont",
             "Massachusetts",
             "New Hampshire",
             "Maine",
             "Rhode Island"]
random_state = random_value(northeast)
```

Help and additional functions

- Python documentation, e.g. https://docs.python.org/3/library/math.html
- StackOverflow e.g. calculate logarithm in python: https://stackoverflow.com/questions/33754670/calculate-logarithm-in-python

```
import math
print(math.sqrt(256))
print(math.log(100))
print(math.log(100)/math.log(10))
```

```
16.0
4.605170185988092
2.0
```

Web Scraping Example: Top Billboard Records from Ultimate Music Database

(Denis)

For a sample of the html for this database, see http://sandbox.data-services.hosting.nyu.edu/ultimate-music-db.html.

```
from bs4 import BeautifulSoup
import csv
from urllib.request import urlopen
import string

url = 'http://www.umdmusic.com/default.asp?Lang=English&Chart=D&ChDate=20171125&ChMode=P'
html = urlopen(url)
soup = BeautifulSoup(html)
```

```
# Alternatively
import requests
r = requests.get(url)
soup = BeautifulSoup(r.text)
```

```
main_table = soup.find_all('table')[9]
```

```
main_table
```

```
from IPython.core.display import display, HTML
display(HTML(str(main_table)))
```

```
records = []
for tr in main_table.find_all('tr')[2:]:
    tds = tr.find_all('td')
    #url = tds[8].a.get('href')
    records.append([elem.text.strip() for elem in tds])
print(records)
```

```
print(records[0])
print(len(records[0]))
```

```
print(records[0][4])
print(len(records[0][4]))
```

```
with open('listing.csv', 'w+', encoding='utf-8') as f:
    writer = csv.writer(f)
    writer.writerows(records)

# here we're essentially done
```

```
# check that the file is in the current directory
import os
os.getcwd()
files = [f for f in os.listdir('.') if os.path.isfile(f)]
files
```

Dealing with Dates

```
# Dealing with dates:
from datetime import datetime, timedelta

start_date = datetime(2017, 11, 25)
d = start_date - timedelta(days=7)
print(d.strftime('%Y%m%d'))
new_url = 'http://www.umdmusic.com/default.asp?Lang=English&Chart=D&ChDate=' + d.strftime('%Y%m%d') + '&ChMode=P'
print(new_url)
```

```
datetime(2017,11,24) < start_date
```

Note the mini-language used to specify date format:

https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior

```
from bs4 import BeautifulSoup
from urllib.request import urlopen
import csv
from datetime import datetime, timedelta
import string

from time import sleep
import random

start_date = datetime(2017, 11, 25)
end_date = datetime(2017,9,1)

d = start_date
records = []
while d > end_date:
    print(d.strftime('%Y%m%d'))
    new_url = 'http://www.umdmusic.com/default.asp?Lang=English&Chart=D&ChDate=' + d.strftime('%Y%m%d') + '&ChMode=P'
    html = urlopen(new_url)
    soup = BeautifulSoup(html)
```

```
    try:
        main_table = soup.find_all('table')[9]

        for tr in main_table.find_all('tr')[2:]:
            tds = tr.find_all('td')
            records.append([elem.text.strip() for elem in tds])

    except:
        print('No data for ' + d.strftime('%Y%m%d'))

    d = d - timedelta(days=7)

    sleep(random.uniform(0.5,2))



with open('listing.csv', 'w+', encoding='utf-8') as f:
    writer = csv.writer(f)
    writer.writerows(records)
```

```
len(records)
```

Challenge Answers

More python resources here

Challenge 1

The correct answer is 2: nothing is True and False at the same time

Challenge 2

- random() gives a floating point value between 0 and 1, not including 1
- If there are 9 students, random() x 9 is a number between 0 and 9.
- len(my_students) gives the number of elements in the list
- int() rounds the number down

```
my_students[int(random()*len(my_students))]
```

Challenge 3

```
def is_palindrome(input):
    if input == input[::-1]:
      return True
    else:
      return False

is_palindrome("tacocat")
```

```
def palindrome_search(paragraph):
  tokens = paragraph.split(" ")
  palindromes = []
  for token in tokens:
    if len(token)>1 and token == token[::-1]:
      palindromes = palindromes + [token]
  return palindromes
```

Challenge 4

```
def vowel_count(x):
  count_vowels = 0
```

```
  for char in x:
    if char in ['a','e','i','o','u']:
      count_vowels+=1
  return count_vowels

vowel_count("Hello World")
```

Challenge 5

```
def random_value(list_in):
  return list_in[round(random()*(len(list_in)-1))]

northeast = ["New York",
             "New Jersey",
             "Pennsylvania",
             "Connecticut",
             "Vermont",
             "Massachusetts",
             "New Hampshire",
             "Maine",
             "Rhode Island"]
random_state = random_value(northeast)
print(random_state)
```