lecture0

July 7, 2020

In [1]:
```
# There are many ways to use python.
# This method (using Jupyter Notebooks)
# runs small pieces of code inside of these rectangular cells.
# Evaluate a cell by holding shift and pressing enter.

# This line is a comment and not a piece of code,
# indicated by the # sign.
```

In [2]:
```
# A fundamental concept in programming
# is the variable. A variable has a name
# and refers to some information, which can vary.

# Here, we assign the variable called my_variable
# a few values, first, the integer 10
my_variable = 10

# the single equals sign "=" does not check equality
# like the "=" in math. Instead, it stores the value
# on its right into the variable on its left
```

In [3]:
```
print("my_variable is:",my_variable)

# we won't explain much about print(...)
# now other than that doing print(...)
# causes ... to be printed to the screen
# print(a,b) prints both a and b.

# quotes are used to distinguish text that should
# be treated as information (to be stored or processed by code)
# from the code itself.
```

```
my_variable is: 10
```

In [4]:
```
# In python, we need to use two equals signs to get
# the traditional mathematical equality
print(3 == 4)
print(3 == 3)
```

```
False
True
```

```
In [5]: # As promised by their name, variables can vary their value.
        # over-write the value 10 with the value "mark"
        my_variable = "mark"
        print(my_variable)

        # variables in python can have any name that starts
        # with a letter and that contains letters, numbers, underscores
        x_is_a_variable_2 = 11
        print(x_is_a_variable_2)
```

```
mark
11
```

```
In [6]: # Once a variable refers to a value,
        # you can treat it as the value itself.
        # There's nothing special about the number 10
        # being inside of a variable.
        x1 = 10
        x2 = 3
        print(10 - 3)
        print(x1 - x2)
        # notice also that python can handle basic math.
```

```
7
7
```

```
In [7]: # store a sequence of numbers in a list
        my_list = [1,2,3,4,5]
        # or any sequence of values
        another_list = [x1,x2,"mark"]
        # lists are denoted by comma separated
        # entries between square brackets [...,...,...]
        print(my_list)
        print(another_list)
```

```
[1, 2, 3, 4, 5]
[10, 3, 'mark']
```

```
In [8]: # you can access an entry of a list
        # by "indexing" into it as follows
        my_list = ["a","b","c","d","e"]
        print("first entry of list is:",my_list[0])
```

```
        # this prints the first thing in the list.
        # In many languages, counting starts at 0.
        # This means my_list[4] is the last element
        # of the list, and my_list[5] is "out of bounds".
        # We will get an error if we un-comment the next
        # line of code:
        # print(my_list[5])
```

```
first entry of list is: a
```

```
In [9]: # Suppose we want to sequentially print each
        # thing in a list. We could do
        print(my_list[0])
        print(my_list[1])
        print(my_list[2])
        print(my_list[3])
        print(my_list[4])
        # but it seems repetative and not CS-like to do.
        # We should learn how to describe to Python that
        # we want some sequential repetative behavior
        # without having to duplicate code ourselves.
```

```
a
b
c
d
e
```

```
In [10]: # introducing the "for-loop"
         for each_item in my_list:
             print(each_item)

         # note on syntax:
         # - "for" and "in" are special python keywords
         # - after "for", we pick a new variable name
         # - after "in" we name an existing list
         # notice the colon and the indentation of the next line
         # - for indentation, always use a single "tab" instead
         # of typing many spaces

         # how it works:
         # The code that's indented will happen several times
         # First, each_item is assigned the first item in the list,
         # my_list[0]. Next time, each_item will become my_list[1]
         # This keeps happening until each_item becomes the last
         # item of the list.
```

```
a
b
c
d
e
```

In [11]:
```
# That is, the above code is shorthand for:
each_item = my_list[0]
print(each_item)
each_item = my_list[1]
print(each_item)
# and so on.
```

```
a
b
```

In [12]:
```
# "each_item" is just a variable name we picked
# we could pick any fresh variable name, like x.
# we can also do more than one thing in the
# body of the loop
my_number_list = [0,1,2,3,4]
for x in my_number_list:
    print(my_list[x])
    print("hello world")
print("end")
```

```
a
hello world
b
hello world
c
hello world
d
hello world
e
hello world
end
```

In [13]:
```
# compare this behavior with
my_number_list = [0,1,2,3,4]
for x in my_number_list:
    print(my_list[x])
    print("hello world")
    print("end")
# So indentation is important.
```

```
a
hello world
end
b
hello world
end
c
hello world
end
d
hello world
end
e
hello world
end
```

```
In [14]: # Suppose we want to do the following loop
         for number in [0,1,2,3,4]:
             print("hello world")
             print("hello world")
             print("hello world")
             print("hello world")
             print("hello world")

         # notice "number" doesn't have to be used, we just print
         # The list is just controling how many times we loop.

         # but now we want to print "hello" 25 times. What can we do?

         # Attempt 1: copy/paste code that print
         # "hello". No thanks.

         # Attempt 2: for number in [0,1,2,3,4,5,6,7,8,9,10,etc...]
         # No thanks
```

```
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
```

```
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
```

```
In [15]: # Attempt 3: nest two loops:
         for number_a in [0,1,2,3,4]:
             for number_b in [0,1,2,3,4]:
                 print("hello world")
```

```
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
```

```
In [16]: # another nested loop example to make a small dance
         for move_one in ['step left','step right','step left']:
```

```
            for move_two in ['say yeah!', 'say no!']:
                print(move_one)
                print(move_two)
        # so move_two loops throug all of its values before
        # move_one changes to each next value.
```

```
step left
say yeah!
step left
say no!
step right
say yeah!
step right
say no!
step left
say yeah!
step left
say no!
```

In [17]:
```
# Attempt 4: in python, range(5) is something similar
# to the list [0,1,2,3,4].

# more generally, range(x) is something similar to the list
# [0,1,2,3,4,...,x-1]

# We can do this, which
# accomplishes what Attempt 2 wanted to:
for number in range(25):
    print("hello world")
```

```
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
```

```
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
```

```
In [18]: # Remember, nothing special about "number",
         # it's just a variable name. We can do
         for x in range(25):
             print("hello world")
```

```
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
```

```
In [19]: # Great, we've learned variables, lists, and loops.
         # One more concept for today
```

```
In [20]: # variables can refer
         # to truth values instead of numbers or strings of characters
         # like "mark". Python has two truth values, True and False
         x = True
```

```
x = False
print(x)
print(3==3)
```

```
False
True
```

In [21]:
```
# Sometimes we want code to do one thing
# or another depending on some condition.

mark_excited = True

if mark_excited == True:
    print(": )")
else:
    print(": (")

# if the condition after "if" is True,
# then then the first thing happens,
# otherwise, the second thing happens.

# try changing mark_excited to False and
# running this code cell again.
```

```
: )
```

In [22]:
```
# we can place an if-else inside of a for-loop
for some_truth_value in [False, False, True]:
    if some_truth_value == True:
        print(": )")
        print("yes")
    else:
        print(": (")
        print("no")
```

```
: (
no
: (
no
: )
yes
```

In [23]:
```
# more generally, programming works by placing small
# pieces of code inside of others.
```

```
# Next time, we will learn about functions,
# which assign several lines of code to a single variable,
# allowing us to easily combine large amounts of code

# Thanks!
```