Tutorial notes 01182021

- Boolean logic and equivalence testing
- If...elif...else control statements

Equivalence testing and booleans

- remember that a single = is the "assignment" operator, so x = y reads "x is assigned y"
- the == syntax is a test for equivalence, so x==y reads "does x equal y?"

In [1]:
```
0 == 0
```

Out[1]:
```
True
```

In [ ]:
```
0 == 1
```

In [ ]:
```
n = 'in'
x = 'in'

print(n==x)
print(n.upper()==x.upper())
```

The "not" operator can be used to see if two expressions are not equivalent

In [ ]:
```
# not operator
n != x
```

Greater than, less than

In [ ]:
```
x = 5
y = 6

x < y
```

logical and, or

- are x AND y both true?
- is x OR y true?

In [ ]:
```
# logical and - will return true if and only if all statement eval to true
x = 5
y = 6
z = 5
w = 8

x < y and z < w and y < w
```

In [ ]:
```
# logical or - will return true if ANY statement evals to true
x = 5
y = 6
z = 5
w = 8

print(x > y or z < w)

# can string together and and or operators...
((x < y) and (z < w)) or (y > w)
```

If...then statements

- test a conditional statement, if true, then do one thing, if false, then do something else...

In [ ]:
```
# simple example using if...else
x = 5
y = 3

if (x < y):
  print('harry wins!')
else:
  print('malfoy wins!')
```

Indentation matters!

- indentation is how Python figures out which statements belong to which part of a if...else structure

In [ ]:
```
# simple example using if...else
x = 5
y = 6

if (x < y):
  print('thor wins!')

else:
  print('hulk wins!')

print('En Dwi wins!')
```

In [ ]:
```

```

If...elif...else syntax

- used to test a series of conditionals

In [ ]:
```
# set up a if...else statement
names = ['shi', 'renita', 'blake', 'vy']

# set up the if...else
# can combine with and and or as well
for n in names:
  if n == names[0]:
    print('hi')
  elif n == names[1]:
    print('hello there')
  elif n == names[2]:
    print('see you')
  else:
    print('who is that???')
```

```
hi
hello there
see you
who is that???
```

Concatenating lists and "lists in lists"

- Use the '+' operator to concatenate lists into a bigger list
- You can make a list of strings or numbers (or both) which is what we've been doing mostly
- You can make a list of lists!

In [1]:
```
# make a few lists
student1 = ['sabina', 19]
student2 = ['valerie', 21]
student3 = ['oleg', 27]
```

Now concatenate the lists into a longer list using the '+' operator

In [2]:
```
concat_list = student1 + student2 + student3

# when you print, you'll see that you now a single list that contains all the names, ages
print(concat_list)
```

```
['sabina', 19, 'valerie', 21, 'oleg', 27]
```

Now make a list of lists!

In [ ]:
```
list_in_list = [student1, student2, student3]
# compare the output here to the output above for 'concat_list' - you'll see that
# list_in_list actually has three sub-lists in it
print(list_in_list)
```

```
[['sabina', 19], ['valerie', 21], ['oleg', 27]]
```

Indexing into a list of lists

- use the [][] to index into lists of lists - the argument in the first set of brackets indicates which sub-list, and the argument in the second set of brackets indicates which element in that sub-list you want

In [ ]:
```
# this will print the first element of the first list in list_in_list
print(list_in_list[0][0])

# this will print the second element of the second list in list_in_list
print(list_in_list[1][1])

# this will print the first element of the third list in list_in_list
print(list_in_list[2][0])

# so...if you want to just increment the ages you can index into the age field in each of the sub-lists and
```

```
sabina
21
oleg
```

Views vs copies

- views share the same data
  - so changing one will change the other
- copies have independent data
  - so changing one will not affect the other

In [ ]:
```
# view vs copy

# first create a list of numbers
x = list(range(0,3))

# create a view of the list
# if you create a view, x and y will be referencing (looking at, referring to) the same data
# here if you modify y, that will also change x
y = x

# should give the same output...
print(x)
print(y)
```

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

Now modify y to see what happens to x

- even though you don't explicitly change 'x', it changes because it refers to the same data as 'y'

In [ ]:
```
print('original x: ', x)
y[0] = 10
print('new x: ', x)
print('new y: ', y)
```

```
original x:  [0, 1, 2]
new x:  [10, 1, 2]
new y:  [10, 1, 2]
```

Copy - independent variable that is separate from the original

- slicing and then re-assigning to a new variable automatically makes a copy, not a view
- can also use the copy() method

In [ ]:
```
# create a copy this time, which will make
# an indepent object y that contains its own
# copy of the data in x
x = list(range(0,3))
y = x[:]  # slicing makes a copy!
# y = x.copy()

y[0] = 10

print(x)
print(y)
```

```
[0, 1, 2]
[10, 1, 2]
```

Tuple object type

- tuples are like lists, but they are immutable.
- use the ()
- can't sort as they are immutable (so no sort method), but can use the general 'sorted()' function

In [ ]:
```
# make a tuple using () (instead of [] for a list)
a_tuple = ('john', 'enrique', 'mariela')

print(a_tuple[1])

# can't modify contents
# a_tuple[0] = 'bob'
# print(a_tuple)

# but you can reassign it.
a_tuple = ('adnan', 'shreya', 'ralph')
print(a_tuple)
```

```
enrique
('adnan', 'shreya', 'ralph')
```

Tuple packing/unpacking

- When writing functions (will learn about soon), you can only return one variable
- however, that variable can be a tuple, allowing you to actually return multiple variables, all packed into one tuple object
- **need to be really careful when unpacking - if you get the wrong variable in the wrong place, it will still work but then you might swap name/age (or whatever your variables are)**

In [ ]:
```
# set up a tuple to store user info
usr_data = ('John', 41, 'CA')
print(usr_data)
```

```
('John', 41, 'CA')
```

In [ ]:
```
# unpack the tuple into the 'pieces'...
(name, age, residence) = usr_data

# bad
# (age, name, residence) = usr_data

print('name:', name)
print('age:', age)
print('residence:', residence)
```

```
name: John
age: 41
```