Lesson 4: Conditional Statements, Loops, and Functions

Key Ideas:

1. Understanding if-else statements
2. Creating new functions; understanding the motivation for creating new functions
3. Understanding for loops; tracing how values change during loops

1. Conditional Statements

We can implement statements of the form "If A, then B; else, C" in python.

Suppose that we would like python to carry out one of two tasks depending on whether a particular condition is satisfied:

- If `CONDITION1` is `True`, then carry out `TASK1`
- If `CONDITION1` is `False`, then carry out `TASK2`

We can accomplish this as follows:

```
if CONDITION1:
    TASK1
else:
    TASK2
```

Note that the indentation here is very important.

Example

Let the variable `age` be someone's age. Depending the value of `age`, we will print out one of two possible texts:

- If `age` is 18 or greater, print out 'You can vote'
- Otherwise (i.e., if `age` is less than 18), print out 'You cannot vote'

In [ ]:
```
age = 21 # experiment and try a few other values


if age >= 18:
    print('You can vote')
else:
    print('You cannot vote')
```

In [ ]:
```
# note that the expression age >= 18 on its own evaluates to True or False:

age = 21 # experiment and try a few other values

age >= 18
```

In general, the "else" clause could be skipped. For example, suppose we don't want to print out anything if `age` is less than 18, we can skip the else clause

In [ ]:
```
age = 17 # experiment and try a few other values


if age >= 18:
    print('You can vote')

# since age is set as 17 above and no else clause is included, nothing is printed out.
```

Boolean operations

Boolean values are values that are either `True` or `False`. Suppose that x and y are boolean values, we can produce a new boolean value using the following operations:

- `and`: `x and y` will have a value `True` when both `x` and `y` are `True`; otherwise, the result is `False`
- `or`: `x or y` will have a value `True` when one or both of `x` or `y` is `True`; otherwise, the result is `False`
- `not`: `not x` will have a value of `True` when `x` is `False`, and will have a value of `False` when `x` is `True`

Please run the code cells below and experiment with the given examples

In [ ]:
```
# and

x = True    # modify this value
y = False   # modify this value
z = x and y

print(z)
```

In [ ]:
```
# or

x = True    # modify this value
y = False   # modify this value
z = x or y

print(z)
```

In [ ]:
```
# not

x = True    # modify this value
y = False   # modify this value
w = not x
z = not y

print(w)
print(z)
```

In [ ]:
```
# combine!

w = True    # modify this value
x = True    # modify this value
y = False   # modify this value

z = (x or y) and (not w)     # try modifying this expression

print(w)
print(z)
```

Example

An online retailer provides free shipping to customers whose order is at least $50 and whose shipping address is in the United States.

For each customer, the online retailer maintains two variables:

- `total_order`: a numerical value that keeps track of the customer's total order
- `has_us_address`: a boolean value (`True` or `False`) whose value is `True` if the order is to be shipped to an address in the United States.

Write an if-else statement that accomplishes the following:

- Prints 'Free shipping!' if the total order is at least 50 and the shipping address is in the United States.
- Prints 'No free shipping :(' otherwise

In [ ]:
```
# example values
total_order = 49.99     # try modifying this value
has_us_address = True   # try modifying this value

if total_order >= 50 and has_us_address :
    print('Free shipping!')
else:
    print('No free shipping :(')
```

Exercise

Consider the same online retailer above. Recall that this retailer provides free shipping to customers whose order is at least $50 and whose shipping address is in the United States.

In addition, the retailer is distributing a promotion code to select customers. Customers who has a promo code will receive free shipping regardless of their order total and regardless of whether their order is to be shipped to a US address.

For each customer, the online retailer maintains three variables:

- `total_order` : a numerical value that keeps track of the customer's total order
- `has_us_address` : a boolean value (`True` or `False`) whose value is `True` if the order is to be shipped to an address in the United States.
- `has_promo_code` : a boolean value (`True` or `False`) whose value is `True` if the customer has entered a promo code.

Write an if-else statement that accomplishes the following:

- Prints 'Free shipping!' if
  - the total order is at least 50 and the shipping address is in the United States, OR
  - the customer has entered a promo code
- Prints 'No free shipping :(' otherwise

In [ ]:
```

```

Nested if-else statements

We can have one if-else statement within another if-else statement.

Example

Let the variable `age` be someone's age and the variable `citizenship` be the person's country of citizenship. Depending the contents of `age` and `citizenship` , we will print out one of two possible texts:

- If `age` is 18 or greater, check the value of `citizenship` :
  - If `citizenship` is equal to 'United States', then print out 'You can vote in US elections'
  - Otherwise, print out 'You cannot vote in US elections'
- Otherwise (i.e., if `age` is less than 18), print out 'You cannot vote in US elections'

In [ ]:
```
age = 21 # experiment and try a few other values
citizenship = 'United States' # experiment and try a few other values

if age >= 18:

    # here is another if-else statement inside the first one
    if citizenship == 'United States':
        # NOTE: two equal signs is needed above because we are checking whether or not the citizeship
is 'United States'
        #       if we only use one equal sign, we will see an error message because python interpret t
hat as a value assignment (try it!)

        print('You can vote in US elections')
    else:
        print('You cannot vote in US elections')

else:
    print('You cannot vote in US elections')
```

If statements with multiple cases

Suppose that we would like python to carry out one of several tasks depending on whether a particular condition is satisfied:

If CONDITION1 is True, then carry out TASK1 Else, if CONDITION2 is True, then carry out TASK2 Else, if CONDITION3 is True, then carry out TASK3 ... Else (if all previous conditions are false), then carry out FINALTASK

We can accomplish this as follows:

```
if CONDITION1:
    TASK1
elif CONDITION2:
    TASK2
elif CONDITION3:
    TASK3
elif ... :
    ...
else:
    FINALTASK
```

Example

Suppose that an instructor in a course wants to provide quick comments on an assignment in an automated way based on student's score (0 to 100), as follows:

- If the score is greater than or equal to 90, then provide the comment 'Excellent work!'
- If the score is less than 90 but is greater than or equal to 80, then provide the comment 'Great job!'
- If the score is less than 80 but is greater than or equal to 60, then provide the comment 'Good progress!'
- Otherwise (if the score is less than 60), then provide the comment 'Please review the topic and see me if you have any questions.'

Suppose we store the assigment score as `score`. We will assign a feedback (let's name it `comment`) depending on the value of `score`, and print out the feedback stored in `comment` at the end.

In [ ]:
```
score = 75 # try other numbers as well
comment = ''  # this is currently an empty text, but we will update it below

if score >= 90 :
    comment = 'Excellent work!'
elif score >= 80 :
    comment = 'Great job!'
elif score >= 60 :
    comment = 'Good progress!'
else :
    comment = 'Please review the topic and see me if you have any questions.'

print( comment )
```

Exercise

Consider the above example. Suppose that the instructor would like to modify their feedback as follows:

- If the score is less than 60 but greater than 50, the comment is 'Please review the topic and see me if you have any questions.'
- But if the score is less than 50, the comment is 'Please see me during my office hours.'

Please modify the code cell below (copied-and-pasted from above) to reflect this change.

In [ ]:
```
# Exercise
# TODO: Modify this code cell

score = 75 # try other numbers as well
comment = ''  # this is currently an empty text, but we will update it below

if score >= 90 :
    comment = 'Excellent work!'
elif score >= 80 :
    comment = 'Great job!'
elif score >= 60 :
    comment = 'Good progress!'
else :
    comment = 'Please review the topic and see me if you have any questions.'

print( comment )
```

2. Functions

We have used many functions in python. For example:

- `np.sum( [1, 2, 3] )` returns the value 6
- `len( [1, 2, 3] )` returns the value 3
- etc.

There are a lot of useful functions in python that we can use. However, sometimes we have a specific task for which we need to write our own python function.

Example

You decided to check out a very popular restaurant in your neighborhood. After waiting in line for two hours, you are finally seated. As you are reading the menu, you realized that this restaurant is cash-only. You have $28.75 with you and need to make sure that you have enough cash to pay for the dinner, including the 8.875\% tax and the tip.

- You are considering ordering a $15 dish and a \$6 beverage. How much would you have to pay if you are giving an 18\% tip?
- Could you afford a $17 dish and a \$6 beverage, with the same 8.875\% tax and 18\% tip?
- Could you afford this meal if you only give a 15\% tip?

The three code cells below help us answer the above three questions.

In [ ]:
```
import numpy as np

order_list = [15, 6]  # this list contains the cost of each item you order
tax_rate = 0.0875     # tax rate; in this example, 8.75%
tip_rate = 0.18       # how much to tip; in this example, 18%

subtotal = np.sum( order_list ) # compute subtotal
tax = subtotal * tax_rate       # compute tax
tip = subtotal * tip_rate       # compute tip
total = subtotal + tax + tip    # compute total

print(total)

if total <= 28.75:
    print('You have enough cash on you')
else:
    print('You do not have enough cash on you')
```

In [ ]:
```
order_list = [17, 6]
tax_rate = 0.0875
tip_rate = 0.18

subtotal = np.sum( order_list )
tax = subtotal * tax_rate
tip = subtotal * tip_rate
total = subtotal + tax + tip

print(total)

if total <= 28.75:
    print('You have enough cash on you')
else:
    print('You do not have enough cash on you')
```

In [ ]:
```
order_list = [17, 6]
tax_rate = 0.0875
tip_rate = 0.15

subtotal = np.sum( order_list )
tax = subtotal * tax_rate
tip = subtotal * tip_rate
total = subtotal + tax + tip

print(total)

if total <= 28.75:
    print('You have enough cash on you')
else:
    print('You do not have enough cash on you')
```

Note that **the computation** process in the above three examples **are exactly the same**; the only parts that are **different** are the cost of the **items ordered** and the **tip rates**.

Let us create a new function to do the above repeated computation more efficiently.

Recall that a function has (1) inputs, (2) carries out an action, and (3) produces an output.

Motivated by the above example, we could create a function where

- The **inputs** are (1) the list of costs of the **items ordered** and (2) the **tip rate**
- The action is to carry out the computation for the total bill given the above inputs
- The output is the total bill.

Creating a new function

```
def FUNCTIONNAME(  INPUTS ):

    ACTION

    return( OUTPUT )
```

Example

Create a function called `calculate_bill` that takes two inputs: `order_list` and `tip_rate` . The function carries out the computation for the total bill given the above inputs. Then, it outputs the total bill.

Then, test that the function works properly by using it.

In [ ]:
```
# define the new function:

def calculate_bill( order_list , tip_rate ):   # inside the parentheses here are the inputs

    # the 'action' / the computation
    tax_rate = 0.0875 # assume that tax rate is 8.875% (a constant; is not up to the person who orders
the food)

    subtotal = np.sum( order_list )
    tax = subtotal * tax_rate
    tip = subtotal * tip_rate
    total = subtotal + tax + tip

    return( total )
```

In [ ]:
```
# check that this works

# first, specify what the inputs are
my_order_list = [21, 4, 2]
my_tip_rate = 0.2

# then, "call" / use the function using the specified input values
calculate_bill( my_order_list , my_tip_rate )
```

In [ ]:
```
# check that this works, example 2

# first, specify what the inputs are
your_order_list = [20, 3]
your_tip_rate = 0.25

# then, "call" / use the function using the specified input values
your_bill_total = calculate_bill( your_order_list , your_tip_rate )  # here we create a new variable t
o store the output value
print( your_bill_total )
```

3. Loops

Loops are used when we want to repeat the same task for each member of a list.

Example

I have three favorite fruits: apples, bananas, and watermelons. For each of my favorite fruit, I would like to declare that I like them:

'I like apples!'

'I like bananas!'

'I like watermelons!'

I typed the above three sentences one by one. This isn't so bad because I only have three favorite fruits, but we can imagine that this gets tedious if I have say 100 favorite items. In addition, this is extremely tedious because the three sentences are essentially exactly the same: 'I like [fruit name]!' except that the fruit names are each of the three items on my list.

Tasks like this can be automated by using a loop.

For Loops

There are different types of loops we can work with, but in this lesson we will learn about just one type: for loops.

Here is the format of for loops:

To repeat `TASK` for each `ITEM` in the list `LIST`

```
for ITEM in LIST :
    TASK
```

Example

Consider the example above:

- The list: the list containing my three favorite fruits: ['apples', 'bananas', 'watermelons']
- The task: To print out the sentence 'I like X!' where X is each item in the above list

The python code:

```
for X in ['apples', 'bananas', 'watermelons'] :
    print('I like ' + X + '!')
```

In [ ]:
```
# try the above code:
for X in ['apples', 'bananas', 'watermelons'] :
    print('I like ' + X + '!')
```

In [ ]:
```
# here is a slightly different way to do the same thing:

favorite_fruits =  ['apples', 'bananas', 'watermelon']

for item in favorite_fruits :   # here we use item instead of X; we can use any name as long as we're
consistent in the next line
    print('I like ' + item + '!') # item here refers to each item in the list favorite_fruits.
```

Exercise

Consider the code cell above.

1. Try adding more items into the list; rerun the code cell. Does it do what you expect?
2. Try changing the name `item` in line 5 to any other name of your choice. Do not change anything else. Why do we see an error?
3. In order to avoid the error you see in Question 2 above, what else do we need to change?

In the above example, the loop was carried out over a list that we specified manually. Sometimes, we need to repeat the task over a larger number of repetitions. The following function is useful.

A useful numpy function: `np.arange()`

To produce an array of values from x to y with stepsize z (including x but excluding y):

```
np.arange( x, y, z )
```

If z is not specified, the default step size is 1.

In [ ]:
```
# example
np.arange(0, 6) # try other numbers
```

In [ ]:
```
# example
np.arange(1, 10.5, 0.5) # try other numbers
```

Note: There are other ways to produce a list of integer values. For example, try:

```
list( range(x, y) )
```

In [ ]:
```
# example
list( range(0, 6) )     # similar to np.arange(0, 6)
```

Example

Use a for loop to display the text:
"1 squared is 1"
"2 squared is 4"
... up to
"20 squared is 400"

That is, for each integer from 1 to 20, we want to display the text ' `x` squared is `x**2` '.

- the list: the list containing integers 1 to 20, which can be produced using `np.arange(1, 21)` (we want to include 20, so we go up to 21)
- the task: to print the sentence ' `x` squared is `x**2` ' for each x in the above list.

In [ ]:
```
# example
for x in np.arange(1, 21):
    x_astext = str( x )   # using str( ) converts the numerical value x into a text/string value
    xsquared_astext = str( x ** 2 )   # using str( ) converts the numerical value x ** 2 into a text v
alue
    print( x_astext + ' squared is ' + xsquared_astext )
```

Exercise

Use a for loop to display the text:
"-2 cubed is -8"
"-1 cubed is -1"
... up to
"11 cubed is 1331"

In [ ]:
```
# your work for the above exercise here
# ...
```

Further References

This lesson covers the basics of if-else statements, function definition, and loops. For more on this topic, see https://docs.python.org/3/tutorial/controlflow.html (https://docs.python.org/3/tutorial/controlflow.html)

Lesson 4 Exercises

Exercise 1 *(if-else statements)*

Suppose that the names `x`, `y`, and `z` contains numerical values.

1. Please write an if-else statement that assign a value to `z` based on the values of `x` and `y`, as follows: Let `z` be whichever is the larger one between `x` and `y`.
2. Try different values for `x` and `y` to check that your if-else statement always assigns the correct value for `z`.
3. In the given markdown cell, please briefly (1-2 sentences) summarize your thought process/reasoning. To what mathematical function does this if-else statement above correspond?

In [ ]:
```
# your work for exercise 1 here

x = 3    # try other values for x and y
y = -10  # try other values for x and y

# write an if-else statement to assign a value for z based on values of x and y
# ...
```

...

Exercise 2 *(functions)*

Write a python function called `even_or_odd` which

- takes one integer value `x` as an input and
- returns as an output a text value:
  - 'even' if `x` is an even integer and
  - 'odd' if `x` is an odd integer.

Note: using an if-else statement along with the remainder operation (`%`) is helpful here.

In [ ]:
```
# your work for exercise 2 here

def even_or_odd( x ):
    # ...
    # ...
    # return ...
```

In [ ]:
```
# once you completed the function definition above,
#    check that it behaves correctly by calling/using the function below

even_or_odd( 25 )   # try out different numbers
```

...

Exercise 3 *(functions, loops)*

Suppose that we have a new function called `myfunction`, defined as follows:

```
def myfunction( x, y ):
    z = x
    for i in np.arange(0,y) :
        z = z + i
    return( z )
```

If we run the command

```
myfunction(2, 3)
```

what value would be returned by this function? Try to first answer this question by reasoning through the given codes, without actually running the code. Then, check if your answer is correct by actually writing the code and running it.

[Your answer + reasoning here]

In [ ]:
```
# your work for exercise 3 here
# check if your reasoning is correct by writing the function in this code cell

def myfunction( x, y ):
    # ...
    # ...

myfunction(2, 3)   # then try other input values as well
```

Exercise 4 *(loops)*

Below, we have a data frame called `us_state_capitals` containing four variables (columns) and 50 rows.

- The column `name` contains the state names
- The column `description` contains the states' capital cities

Run the following code cell to load and preview the data frame.

In [ ]:
```
# for exercise 4,
# simply run this code cell to view the contents of the dataframe `state_capitals`
# do not modify this code cell

us_state_capitals = pd.read_csv('https://raw.githubusercontent.com/jasperdebie/VisInfo/master/us-state
-capitals.csv')

us_state_capitals # preview the data frame
```

Your tasks in Exercise 4:

1. What is the output of the following command?

```
us_state_capitals['name'][2]
```

2. What is the output of the following command?

```
us_state_capitals['description'][2]
```

3. What is the output of the following command?

```
state = us_state_capitals['name'][2]
cap = us_state_capitals['description'][2]
print('The capital of ' + state + ' is ' + cap)
```

4. Consider the three lines of codes in Question 3 above.

   - What is the significance of the number `2` inside the pairs of square brackets?
   - Copy and paste the three lines of codes in Question 3, and change the number `2` to `14` ; what do you see?
   - Try changing the number `2` to `55` . What happens?

5. Notice that the capital of Connecticut is listed as 'Hartford\<br\>', which is correct but could be cleaned up by removing the '\<br\>' portion of this text. We can do this by running the following command:

```
us_state_capitals['description'][...] = 'Hartford'
```

   What number should replace `...` ?

   Similarly, please clean up the text for the capital of Georgia by replacing 'Atlanta\<br\>' with just 'Atlanta'.

6. Write a for loop that prints out the sentence of the form 'The capital of [state name] is [capital city name]' for each US state.

In [ ]:
```
# your work for exercise 4 here
# ...
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

...