CMM201 Week 4

1 CONDITIONAL STATEMENTS IN PYTHON

1.1 Aims of the Lecture

- Learn and understand the different conditional statements contained in Python.
- Exemplify practical uses of each statement.
- Learn how data types and data structures interact with these conditional statements

1.2 Additional Reading and Sources

- w3schools
- Real Python

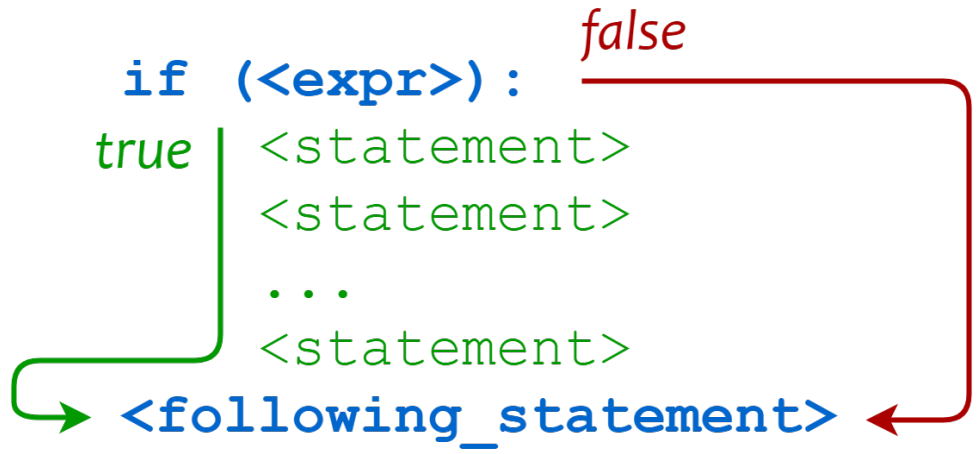
2 The IF/ELIF/ELSE Conditional Statement

- In its most basic form, an *if* statement establishes a condition which, if met, executes the statement.
- We use colon (:) to finish the if condition(s).
- Then, we establish the statement using TAB/INDENT.

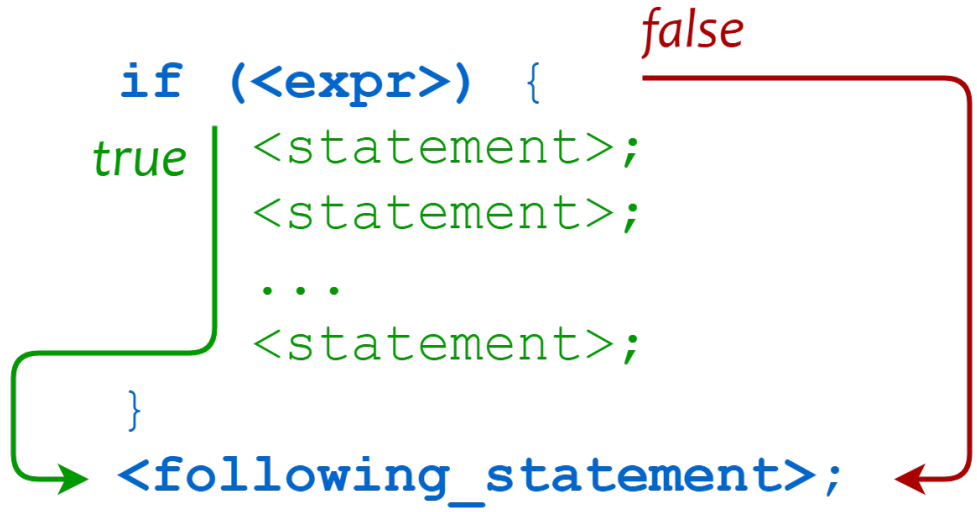
```
In [ ]: # example of an if statement
        if 1<2:
            print('True!')
```

```
In [ ]: # example of an if statement with two conditions
        if 1<2 and 'P' in 'Python':
            print('Also true!')
```

- Python is all about indentation (off-side rule).
  - Coined after football!
- You need to be careful regarding how statements work!
- This is how Python does it.



- This is how other languages (i.e. C++/Java/Perl) do it.




```
In [ ]: # Example
        if 3>2:
            print('This gets done...')
            print('...this also gets done...')
        print('...and this too!')
```

```
In [ ]: # Example (inverted)
        if 3<2:
            print('This gets done...')
            print('...this also gets done...')
        print('...and this too!')
```

```
In [ ]: # Does line execute?                    Yes    No
        #                                      ---    --
        if 'a' in ['a', 3.4, True]:          #  x
```

```
        print('Outer condition is true')      #  x

        if 10 > 20:                           #  x
            print('Inner condition 1')        #        x

        print('Between inner conditions')     #  x

        if 10 < 20:                           #  x
            print('Inner condition 2')        #  x

        print('End of outer condition')       #  x
    print('After outer condition')            #  x
```

- If you want to test a condition, but after being false test another one, use *elif*.
- This has to go aligned with the *if*.

```
In [ ]: if 10>20:
            print('nope')
        elif 10<20:
            print('yep')
```

- If the conditions are not met, then we use *else*.

```
In [ ]: if 3<2:
            print('Unreal!')
        else:
            print('This is the case')
```

2.1 One-line Conditional Statements and Expressions

- Python allows to write conditional statements in one line of code.
- It does not make any difference in computational terms.
- Some people find it more practical.

```
In [ ]: if 1>0: print('Yes'); print('sure')
```

- Multiple statements can also be written.

```
In [ ]: x = 2
        if x == 1: print('a'); print('b'); print('c')
        elif x == 2: print('d'); print('e')
        else: print('f'); print('g')
```

- Conditional (or ternary) operations are also supported in Python.
- This has been proposed in other programming languages to "simplify" syntax.
- In this case you have to use the following convention:
  - (statement_if) if (condition) else (statement_else)

```
In [ ]: # Let's try to run this code changing the rain to True and False
        raining = False
        print("Let's go to the", 'library' if raining==True else 'beach')
```

How would we write this in the usual way?

```
In [ ]: raining = False
        if raining:
            print("Let's go to the library")
        else:
            print("Let's go to the beach")
```

- If you want to define a string with an apostrophe, enclose it in parenthesis.
- ...and vice versa!
- If you want to prove that something is true, you can explicitly say so OR just put the name of the variable

2.2 Pass

- Sometimes you DON'T want to do anything if a condition is met.
- In any other languages, you leave the content of the curly brackets blank.
- Python wouldn't understand that (as there is no such thing).
- Therefore, we need to explicitly tell state in the code that we want to pass.

```
In [ ]: # Let's say you don't want to do anything if 1 equal 1
        if 1==1:

        print('rest')
```

```
In [ ]: # Let's say you don't want to do anything if 1 equal 1
        if 1==1:
            pass
        print('rest')
```

3 For loops

- This is the most elemental method to iterate over a specified range
- Works well with lists, ranges, etc.
- for x in y

```
In [ ]: # print numbers in a range
        for i in range(10):
                print(i)
```

```
In [ ]: # print elements in a list
        basket = ['banana','apple','grape']
        for fruit in basket:
                print(fruit)
```

You can also do calculations while looping.

```
In [ ]: # loop over a list of strings and calculate their length
        words = ['Robert','Gordon','University']
        for w in words:
            print(w,len(w))

In [ ]: # loop over a list of strings and print their INDEX
        for i in range(len(words)):
            print(i, words[i])
```

- Notice that we had to calculate the length of *words*, then produce a range of such length and then iterate the range.
- Instead of doing this, we can use the *enumerate* function.
- In this case you use two variables: the *index* and the *element*.

```
In [ ]: # loop over a list of strings and print their INDEX
        for i, word in enumerate(words):
            print(i, word)
```

Remember that strings are also *mutable* objects containing elements (i.e. letters) and thus they can also be iterated.

```
In [ ]: for i, x in enumerate("banana"):
          print('The letter '+str(i)+' is '+x)
```

- Notice how the index can be converted into a string using the *str()* function.
- Then, it can be appended to the printed string.
- This comes very handy when we want to change the message given to a user depending the iterations/variables.

- But wait! People rarely identify things in the position "0".
- We usually start counting things by one!
- How can we change the previous code to reflect this?

```
In [ ]: for i, x in enumerate("banana"):
          print('The letter '+str(i+1)+' is '+x)
```

3.1 For + IF

- These two are typically used in conjunction.
- For instance, you can loop and find certain elements in a list.

```
In [ ]: # finding certain elements in a list
        basket = ['banana','apple','orange','grape']
        for fruit in basket:
            if 'p' in fruit:
                print(fruit)
```

3.2 The Break

- This instructions lets you get out of a for loop.
- Typically used along with if.

```
In [ ]: # finding certain elements in a list
        basket = ['banana','apple','orange','grape']
        for fruit in basket:
            if 'p' in fruit:
                print(fruit)
                if fruit == 'apple':
                    break
```

3.3 Continue

- We use this statement when we want to stop the current iteration of the loop, BUT we don't want the next instruction to be done.

```
In [ ]: # finding certain elements in a list
        basket = ['banana','apple','orange','grape']
        for i,fruit in enumerate(basket):
            if 'p' in fruit:
                print(fruit)
                if fruit == 'apple':
                    continue
            print(i)
```

```
In [ ]: # finding certain elements in a list
        basket = ['banana','apple','orange','grape']
        for i,fruit in enumerate(basket):
            if 'p' in fruit:
                print(fruit)
                if fruit == 'apple':
                    continue
                print(i)
```

3.4 Nested Loops

- This is a very commonly used resource that lets you loop more than once.
- For instance, you can verify all elements of a matrix, an image, a coordinate plane, a table, etc...
- You can also use this technique to find all combinations between two data structures.

```
In [ ]: adjective = ["red", "big", "tasty"]
        fruits = ["apple", "banana", "cherry"]

        for x in adjective:
            for y in fruits:
            print(x, y)
```

4 While

- This statement is also used as a loop, but it keeps going until a condition is not met.

```
In [ ]: x = 0
        while x<7:
            print(x)
            x=x+1
```

- Be very careful when establishing the stop condition, as you don't want to enter an endless loop!
- In this case, you will notice that the cell doesn't stop.
- You can go to Kernel -> Interrupt to stop the execution.

```
In [ ]: x=0
        while x>=0:
            print(x)
            x=x+1
```