Intermediate python - Control flow structures - 1

One should look for what is and not what he thinks should be. (Albert Einstein)

Control Flow: Topic introduction

In this part of the course, we will cover the following concepts:

- Control flow structures and the practice of writing of modular code
- Conditional statements and blocks
- For loops, while loops, and list comprehensions

Chat question

- In this module, we will explore basic control flow structures in Python including **conditionals** and **loops**
- Before beginning, let's look at some examples of conditional and looping situations you may experience in your everyday life
- Which of these situations represents **conditional** logic?
- **Situation A**: You drive to work every day in your car.
- **Situation B**: If you are at least 16 years old, you can drive a car in the U.S. Otherwise, you are not allowed to drive a car in the U.S.

Chat question

- Which of the situations below represents a **loop**?
- **Situation A:** Your alarm clock is set for every morning at 7:00.
- **Situation B:** If it's Saturday or Sunday, your alarm clock is off. All other days, your alarm clock is set for 6:30 in the morning.

Chat question

- Which of the situations below represents **conditional** logic?
- **Situation A:** If you're really hungry, you eat eggs, bacon, and toast for breakfast. Otherwise, you eat a bowl of cereal.
- **Situation B:** You eat a bowl of cereal every day for breakfast.

Chat question

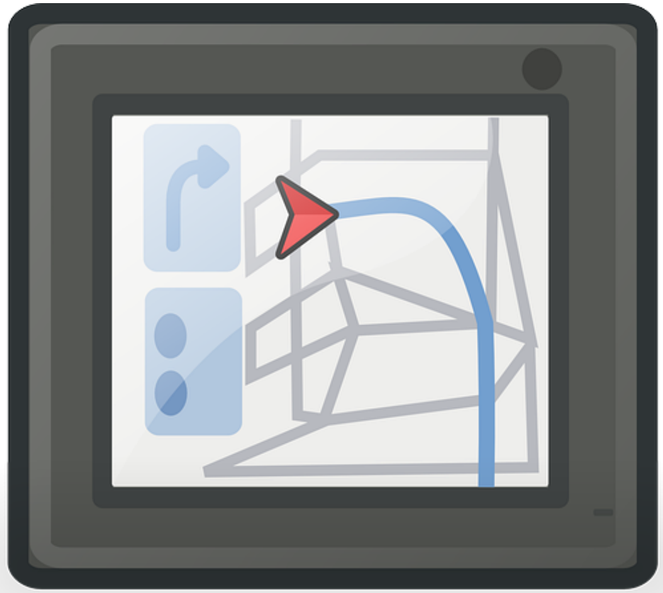
- **Use the microphone or chat** to share your ideas about the following questions:
  - Were there any patterns you noticed?
  - What words indicated a conditional statement?
  - What words indicated a looping statement?

Module completion checklist

Objective	Complete
Discuss control flow structures and the practice of writing modular code	
Use conditional statements such as if / else	

Getting from point A to point B

- When writing code, we give the computer a set of directions just like GPS software provides us when are driving
- This set of directions is referred to as a **control flow**



Control flow

- A **control flow** is a set of directions for a computer program
- Control flows account for different options and lead the program to different outcomes, depending on certain inputs conditions
- We let the computer go through the program (sometimes repeating actions) until it reaches its end or we interrupt the program

Control flow structures

- The directions that allow us to control the flow of a program are called **control flow structures** in programming languages
- Control flow structures are very similar across languages in their concept… (but not syntax!)

Basic control flow structures in Python

- We will discuss the following control flow structures:
  - **Conditionals** (`if-else` statements)
  - **Loops** (`for` and `while` structures)
  - **Functions** (`def` structure)

Control flow structure capabilities

- Control flow structures allow us to:
  - Point the program in the right direction *(like conditional statements)*
  - Perform the same tasks multiple times without writing out every step *(like loops)*
  - Abstract out some actions that can be re-used later by another part of the program or another program all together *(like functions)*
  - Save **time and space**
  - Make our programs **clean and readable**
  - Make our code **modular and re-usable**

Module completion checklist

Objective	Complete
Discuss control flow structures and the practice of writing modular code	✔
Use conditional statements such as if / else	

Conditional statements

- ***Conditional statements*** allow our program to decide whether or not to run certain sections of code, based on some condition
  - If the condition is **true**, the program takes one turn
  - If the condition is **false**, the program takes another turn

- In Python, we use `if` and `else` blocks to build *conditional statements*
- The *condition* itself follows the `if` statement
  - The outcome if the condition is **true** follows on the next line
- The *alternative* route is given by the `else` block
  - The outcome if the condition is **false** follows afterward

Conditional statements (cont'd)


```
if 2 + 2 == 4:
    print('Good, we are sane!')
else:
    print('We are living in 1984!')
```

```
Good, we are sane!
```

Recap: Logical operators

- The most common way to create Booleans (i.e. **true** or **false** values) is by using *logical operators*

Operator	Example
Greater than	x > y
Less than	x < y
Equal to	x == y
Not equal to	x != y
Greater than or equal to	x >= y
Less than or equal to	x <= y

Recap: Combining Booleans

- We can use `and` and `or` to check for a combination of conditions

```
# Check if 2 conditions are true
# for the expression to return `True`!
x = 8
print(x > 5 and x < 10)
```

```
True
```

- We can use `add` to add as many statements as we want, but when Python checks the conditions, it will only return `True` if **all** of the conditions are met

```
# Every single condition much be true for this expression to return `True`!
print(x > 5 and x > 1 and abs(x) == 7 and x < 10)
```

```
False
```

- Since `abs(x)` is equal to `8` not `7`, the entire expression returns `False`

Condition types

- Conditions vary in many ways, but they absolutely need to result in a **true** or **false** output
- Conditional expressions (e.g. `2 + 2 == 4`) can also be assigned to variables
- Long conditional statements are often more readable that way within `if-else` blocks

```
condition = 2 + 2 == 4
print(condition)
```

```
True
```

```
another_condition = "this string" == "that string"
print(another_condition)
```

```
False
```

```
yet_another_condition = 5 + 10 > 10 + 5
print(yet_another_condition)
```

```
False
```

Condition types (cont'd)

- Conditional statements can be compound (i.e. consist of multiple conditions that result in a single **true** or **false** output)

```
compound_condition1 = (5 + 10 > 10 + 5) and ("this string" == "this string")
print(compound_condition1)
```

```
False
```

```
compound_condition2 = (5 + 10 >= 10 + 5) and ("this string" == "this string")
print(compound_condition2)
```

```
True
```

```
# Here, it's helpful to look at each condition individually.
# If one of them is true, then the whole statement is true.
compound_condition3 = (compound_condition1 and compound_condition2) or (100/2 > 100 % 2)
print(compound_condition3)
```

```
True
```

Putting it all together

- Here is an `if-else` statement with a compound condition written directly within it

```
if (5 + 10 > 10 + 5) and ("this string" == "this string"):
    print('Compound condition 1 is true, do something!')
else:
    print('Compound condition 1 is false, do something else!')
```

```
Compound condition 1 is false, do something else!
```

- Now here is the same `if-else` statement with the compound condition saved to a variable beforehand

```
compound_condition1 = (5 + 10 > 10 + 5) and ("this string" == "this string")

if compound_condition1:
    print('Compound condition 1 is true, do something!')
else:
    print('Compound condition 1 is false, do something else!')
```

```
Compound condition 1 is false, do something else!
```

- Both ways are correct

Special cases of conditional blocks

- Conditional blocks of code don't have to be in the `if-else` form
- Sometimes you want to take action if the condition is **true**, but don't need to do anything if it is **false**
- In that case, just a single `if` block will do the trick!

```
if compound_condition2:
    print("Ok, I guess I have to do something after all!")
```

```
Ok, I guess I have to do something after all!
```

```
if yet_another_condition:
    print("This means the `compound_condition3` is true, otherwise you will get nothing!")
```

- In the case of the first example, the action is triggered, because compound_condition2 is **true**
- In the second example, the `print` statement is not triggered because `yet_another_condition` is **false**, and we gave our program no alternative action to perform

Special cases of conditional blocks (cont'd)

- If we want to check **multiple conditions**, we can use `elif`
- As soon as the program finds one condition that is satisfied, it will stop running down the `elif` tree
- Suppose you wrote a program to help you make a decision about purchasing a car based on its price

```
price = 37000

if price > 40000:
    print("That's too expensive!")
elif price > 34000:
    print('A little pricey but maybe worth it...')
elif price > 26000:
    print("This seems like a fair price for the quality")
elif price > 22000:
    print("What a good deal! I'll get it")
else:
    print("Hmmm this is pretty cheap, maybe there's a problem with it.")
```

```
A little pricey but maybe worth it...
```

Special cases of conditional blocks (cont'd)

- Sometimes we have a complex set of decisions to make based on the outcome of the other set of decisions
- In that case, **nested** conditionals are the way to go

```
if condition:
    if another_condition:
        print("You got to a nested statement!")
    else:
        print("You still got to the nested statement!")
else:
    print("No luck printing a nested statement!")
```

```
You still got to the nested statement!
```

Actions within conditional blocks

- The last element of conditional blocks are actions
- Virtually anything can go into actions associated with `if` and `elif` blocks

```
price = 37000
account_balance = 45000

if price > 38000:
    action = "Leave the dealership immediately, this is a rip off!"
    account_balance = account_balance - price
elif price > 22000 and price <= 38000 :
    action = "Take the car and go celebrate, you can afford it!"
    account_balance = account_balance - price
else:
    action = "Leave the dealership immediately, this is a scam!"
    account_balance = account_balance - price

print(action)
```

```
Take the car and go celebrate, you can afford it!
```

```
print("Current account balance:", account_balance)
```

```
Current account balance: 8000
```

Actions within conditional blocks (cont'd)

- Let's change `price` and print results

```
price = 41000
account_balance = 45000

if price > 38000:
    action = "Leave the dealership immediately, this is a rip off!"
    account_balance = account_balance - price
elif price > 22000 and price <= 38000 :
    action = "Take the car and go celebrate, you can afford it!"
    account_balance = account_balance - price
else:
    action = "Leave the dealership immediately, this is a scam!"
    account_balance = account_balance - price

print(action)
```

```
Leave the dealership immediately, this is a rip off!
```

```
print("Current account balance:", account_balance)
```

```
Current account balance: 4000
```

Conditionals and actions based on them

- Any action can be housed within a conditional block
- Actions we might take include:
  - Performing mathematical operations
  - Assigning variables new values
  - Building other conditional blocks within them (a.k.a. *nested conditional blocks*)
  - Adding *loops*
  - Calling *functions* or even other *programs*

Knowledge check

Link: ***https://forms.gle/nKTtQeN1iTrsf2Pi8***

Module completion checklist

Objective	Complete
Discuss control flow structures and the practice of writing modular code	✔
Use conditional statements such as if / else	✔

Congratulations on completing this module!

You are now ready to try Tasks 1-4 in the Exercise for this topic