Lecture 2B—Logic flow

In this notebook, we learn to control the logic flow. We will use `if`, `while`, `for` to create condition and iteration.

Table of Content

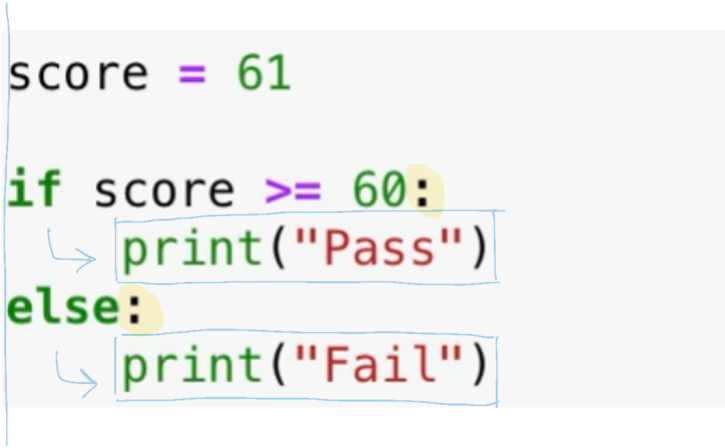
if-condition

In [1]:
```
score = 61

if score >= 60:
    print("Pass")
else:
    print("Fail")
```

```
Pass
```

Let's take a deeper look of the above code block.



Whenever we have logic flow that have code only execute under certain condition, we add a `:` and **indent** the following block.

The **indentation** means the code *belongs to* the condition. In this example, the `print("Pass")` only execute when the condition `score >=60` meets. We can tell from the code that the line of code has **4 leading spaces**.

The `print("Fail")` only execute when the condition `score >=60` fails. We can tell the line of code has **4 leading spaces** under `else:` block.

In the following example, we take the input and compare the input. Remember that the input value is always string. We need to type cast the input value into `int` by using `int()` function.

In [2]:
```
age = input("What is your age? ")

if int(age) >= 18:
    print("You may drink, a little bit.")
else:
    print("Please don't drink.")

print("Good bye and have a nice day.")
```

```
What is your age? 23
You may drink, a little bit.
Good bye and have a nice day.
```

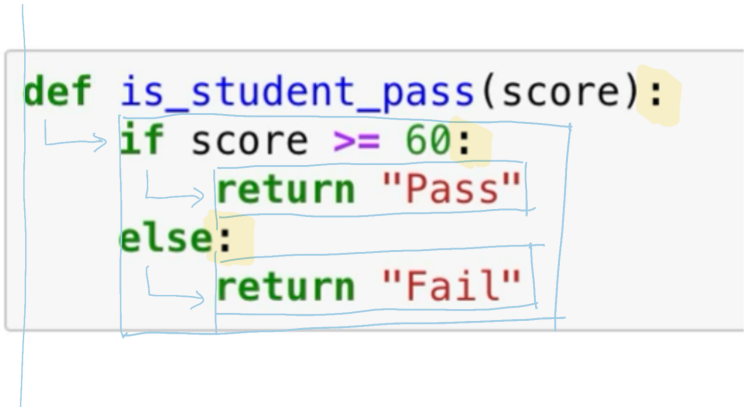
Putting logic into function

Sometimes, the conditional checking repeats for the whole data set. We can define function to reuse the same code. For example, the following code defines a function the *return* either "Pass" or "Fail" based on the given *score* value.

Function is a block of code that takes **parameters** and **return** calculation result. This block of code is executed only when other line of code calls it.

In [3]:
```
def is_student_pass(score):
    if score >= 60:
        return "Pass"
    else:
        return "Fail"
```

Note the **indentation** in the above code. The whole `if` block has at least **4 leading spaces** to indiciate that block of code is under the `def is_student_pass(score):` function. The two `return` statements has 4 further leading spaces to indicate that they are under the `if score>=60:` and `else:` statement.



In [4]:
```
result_a = is_student_pass(59)
print(result_a)
```

```
Fail
```

In [5]:
```
result_b = is_student_pass(61)
print(result_b)
```

```
Pass
```

You can think of function as a block of coded pre-defined and stored under a name. It is like variable for a list of "procedures". Every time we call the function, it executes the pre-defined lines of code and **return** the calculated result.

while-loop vs. for-loop

While-loop is usually used for iteration that we don't know the total count.

For-loop is when we know the iteration count. For example, we already have the list of items. Or we already know how many times to repeat.

for-loop

In [8]:
```
data = ['Apple', 'Banana', 'Orange']

for fruit in data:
    print(fruit)
```

```
Apple
Banana
Orange
```

Let's bring back our `is_student_pass` function we defined. We can use a for loop to get a list of Pass/Fail result.

In [9]:
```
def is_student_pass(score):
    if score >= 60:
        return "Pass"
    else:
        return "Fail"

list = [80, 59, 62, 70, 99, 40, 51]

for score in list:
    result = is_student_pass(score)
    print(f"{score}: {result}")
```

```
80: Pass
59: Fail
62: Pass
70: Pass
99: Pass
40: Fail
51: Fail
```

Exercise: for-loop

In this exercise, we bring back the book lists and loop books to print each details.

In [10]:
```
books = [
    {'title': 'Python Tricks', 'category': 'Programming', 'price': 240},
    {'title': 'Python Crash Course', 'category': 'Programming', 'price': 2(
    {'title': 'Getting Real', 'category': 'Startup', 'price': 200}
]
```

Try to loop the books and print the title and their price. We don't need to print the category in this exercise.

In [11]:
```
for book in books:
    # Your code here
    title = None
    price = None
    print(None)
```

```
None
None
None
```

Expected result
Python Tricks: $240
Python Crash Course: $200
Getting Real: $200

range(start, end, step)

Sometimes, we don't have the list to iterate. But we know how many steps we want to loop. In this case, we can use `range` .

In [12]:
```
# print("0-9")
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [13]:
```
# print("1-10")
for i in range(1,11):
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

In [14]:
```
# print("1,3,5,7,9")
for i in range(1,10,2):
    print(i)
```

```
1
3
5
7
9
```

In [15]:
```
# print("2,4,6,8,10")
for i in range(2,11,2):
    print(i)
```

```
2
4
6
8
10
```

Code example: while-loop

By combining list and while loop, we can keep appending the value to the `tasks` list until the input is "q". The `break` command exits the while-loop.

In [6]:
```
tasks = []

while True:
    value = input("Please input a to-do task, or 'q' to quit. ")

    if value == 'q':
        break

    tasks.append(value)

print(tasks)
```

```
Please input a to-do task, or 'q' to quit. q
[]
```

You can tell from the indentation that line 4—9 under the `while True:` loop. Line 7 is under the `if value=='q':` loop.

Keep in mind that we need to be careful when using `while True:` . the condition is always true so the loop will run forever until we meet the `break` command. If the logic we designed has flaw, the loop may never end.

Exercise: while-loop

In this exercise, we would like to create a guest list by modifying `while` loop example given above.

In [7]:
```
guests = []

# Your code here
while None:
    value = input("Your text here ")
    if value == 'q':
        break

    None # append the value to guests

print(guests)
```

```
[]
```

Summary

In this lesson, we learned the basic logic flow. Specifically:

- if-condition
- loop with or without known iteration count
- while-loop
- for-loop
- range

In [ ]: