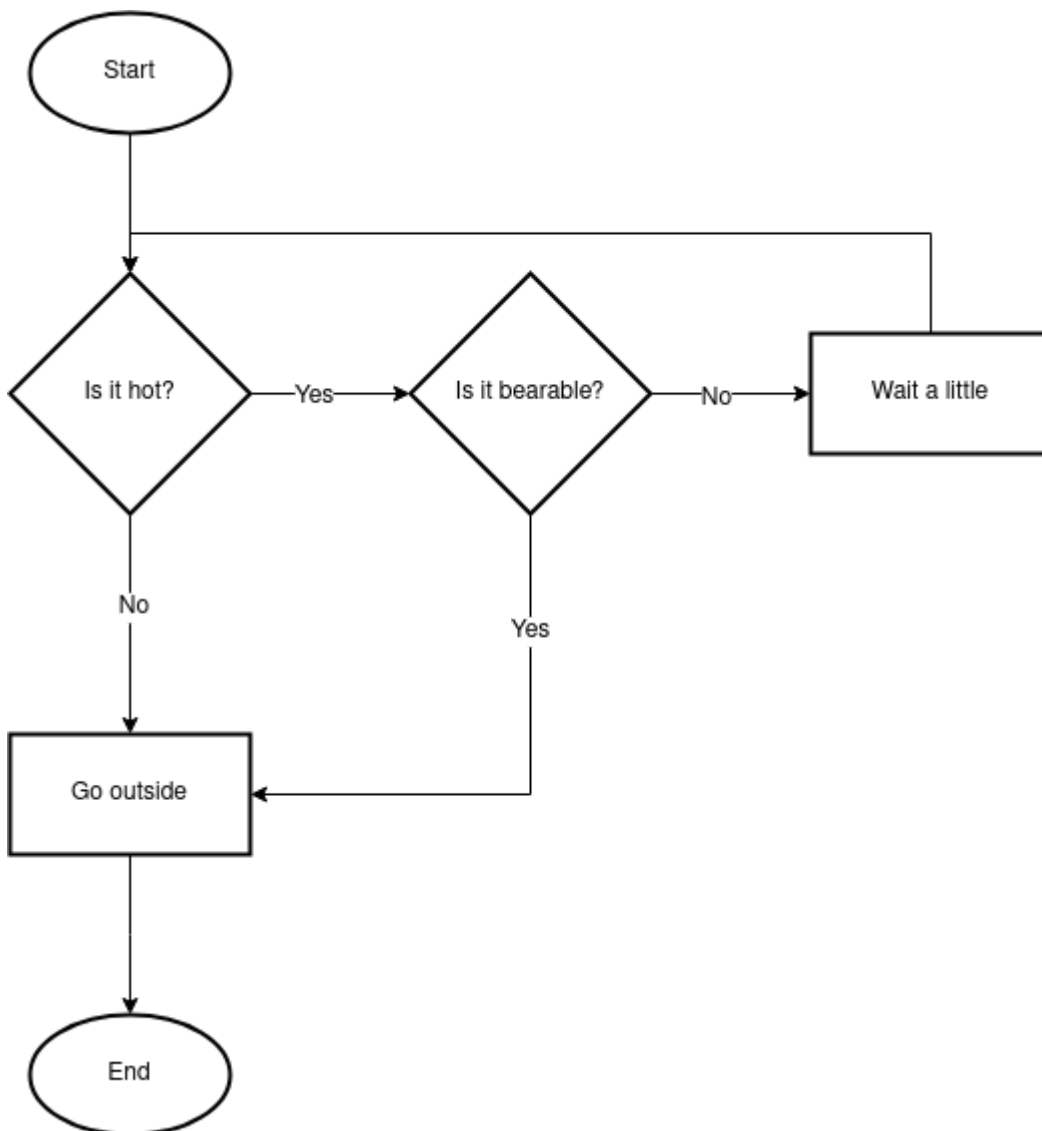


Chapter 2

Chapter 2: Flow Control

When we talk about flow control, we are referring to the ability of a program to decide what to do or how many times to do it, depending on a condition. Take a look at the flow chart below:



In this example, before deciding what to do, we evaluate certain conditions. Is it hot? if it isn't, we go outside. In case it is, we check if it's bearable, if it is, we go outside, if not, we wait a little and check again.

In this chapter you'll learn how to utilize flow control to improve the functionalities of your programs.

Comparison Operators

Before we begin to utilize flow control in our code, we need to talk about comparison operators. They're used to form expressions, which are simply a combination of operators and operands, resulting in a value. We have used expressions previously, for example:

```
number = 10 + 2
```

10 + 2 is an expression that forms the value 12. Nothing new in the horizon. With comparison operators we can create expressions a little more complex. As we can see below:

```
is_less_than = 2 < 10
# is_less_than = True
```

In this expression, python will check if two is less than 10 and the result will be a `boolean`, in other words, true or false.

The comparison operators are:

Operator	What it does
==	equal to
!=	not equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

```
number1 = 10
number2 = 2

print(f"Is {number1} less than {number2}? {number1 < number2}")
print(f"Is {number1} greater than {number2}? {number1 > number2}")
print(f"Is {number1} equal to {number2}? {number1 == number2}")
print(f"Is {number1} not equal to {number2}? {number1 != number2}")
```

```
Is 10 less than 2? False
Is 10 greater than 2? True
Is 10 equal to 2? False
Is 10 not equal to 2? True

Process finished with exit code 0
```

Note: Do not confuse the assign operator (=) with the equality operator (==).

If / Elif / Else

The first form of flow control we'll see is the `if / else`. The idea is simple: if a condition is true, do this, otherwise, do that.

```
name = "Jorge"

if name == "Jorge":
    print("Hello, Jorge")
else:
    print("Hello, person")
```

In this example, we compare if the value of the variable `name` is the equal to "Jorge", python evaluates this expression as true, then executes the following commands. Resulting in:

```
Hello, Jorge

Process finished with exit code 0
```

If the value of the variable `name` was not equal to "Jorge", python would execute the commends after `else`.

Okay, what about the `elif`? The `elif` is utilized when we have more than two options.

```
if name == "Jorge":
    print(f"Hello, Jorge")
elif name == "Janine":
    print("Hello, Janine")
else:
    print(f"Hello, person")
```

First python will check if the value of the variable `name` is equal to "Jorge"; If it isn't, then it checks if the value is equal to "Janine"; If it also isn't, then it will execute the code after `else`.

Note: Remember that the code is executed from top to bottom. Once a condition is true, python will execute the code in its block and will not verify if any other condition is also true.

Important: `if` does not need an `else` statement for it to work. You can have an `if` statement by itself and if the condition is not true, python skips the `if` and continues running the rest of the code. The statement can also be ended with an `elif` instead of an `else`.

Indentation

indentation is the space in the beginning of a line. In most languages, it only exists to organize and beautify the code. In other languages, like python, it is extremely important because it defines where a block of code begins and where it ends. The indentation is created with 4 spaces. Most text editors and IDEs allow us to indent with the *tab* key.

```
if name == "Jorge":  
    # this code is indented and it belongs to the block of code of  
    "if"  
    print("Hello, Jorge")  
else:  
    # this code is indented and it belongs to the block of code of  
    "else"  
    print("Hello, person")
```

A block of code may contain other blocks of code, as we can see below:

```
username = "Ushi"  
password = "litterbox"  
  
if username == "Ushi":  
    print("Hello, Ushi")  
  
    if password == "litterbox":  
        print("Login successful.")  
    else:  
        print("wrong password.")  
  
else:  
    print("unknown user.")
```

Let's see, step by step, how python will execute this code:

1. First, python will check if `username == "Ushi"`, if it is, it will run the following block of code:

```
username = "Ushi"
password = "litterbox"

if username == "Ushi":
    print("Hello, Ushi")

    if password == "litterbox":
        print("Login successful.")
    else:
        print("wrong password.")
else:
    print("unknown user.")
```

2. So it will run `print("Hello, Ushi")` and check if `password == "litterbox"`, if it is, it will run the block of code shown below

```
username = "Ushi"
password = "litterbox"

if username == "Ushi":
    print("Hello, Ushi")

    if password == "litterbox":
        print("Login successful.")
    else:
        print("wrong password.")
else:
    print("unknown user.")
```

3. In case `password` is not "litterbox", it will execute the following block of code:

```
username = "Ushi"
password = "litterbox"

if username == "Ushi":
    print("Hello, Ushi")

    if password == "litterbox":
        print("Login successful.")
    else:
        print("wrong password.")
else:
    print("unknown user.")
```

4. If `username` isn't "Ushi", it will execute the following block of code:

```
username = "Ushi"
password = "litterbox"

if username == "Ushi":
    print("Hello, Ushi")

    if password == "litterbox":
        print("Login successful.")
    else:
        print("wrong password.")
else:
    print("unknown user.")
```

Line breaks

Unlike indentation, which is so important in python, line breaks are often ignored and do not affect your program in any way. Because of this, we can use it to better organize our code, making it more readable.

Logical Operators

Logical operators are used when we want to evaluate two or more expressions. There are 3 logical operators in python: `and`, `or`, `not`.

Note: To help with readability, some people prefer to put the expressions between parenthesis.

and

When we use the `and` operator, python will only run the block of code if all the expressions are true.

```
first_number = 15
second_number = 6

if (first_number > 10) and (second_number < 20):
    print("yay")
```

Here, `first_number` is greater than 10 and `second_number` is less than 20. Both the expressions are true; thus python will run the block of code.

or

When we utilize the `or` operator, python will run the block of code if one of the expressions is true.

```
first_number = 15
second_number = 6

if first_number > 10 or second_number < 5:
    print("yay")
```

In this case, `first_number > 10` is true and `second_number < 5` is false. Python will execute the block of code because one of the expressions is true.

not

When we use the `not` operator, we reverse the boolean value. If the expression is true, it will then become false; if it's false, it will become true.

```
number = 6

if not number < 5:
    print("yay")
```

In this example, `number < 5` is false. Since we're using the `not` operator, the block of code will be executed because `not False` is `True`.

Ternary Operator

The ternary operator is a "one-line `if`", it's a more concise way of writing a simple `if/else`.

```
price = 999.99
can_afford = "no" if price >= 1200 else "yes"
```

The value of the variable `can_afford` will be "no" if the price is greater than or equal to 1200, otherwise, it will be "yes".

Exercise 3

1. Make a program that will ask the user for the grades of 3 exams, calculate the average grade, print it and if the grade is greater or equal to 6, print "You've passed! (:", if it was between 4 and 6, print "You will go to summer school", if it's less than 4, print "you've failed".
2. Make a program that asks the user for their weight and height, calculate their BMI (Body Mass Index), print it with 2 decimal points and print the information following the table below:

BMI	Classification
<18,5	Underweight
18,5 a 24,9	Normal weight
25,0 a 29,9	Overweight
>30,0	Obesity

3. Make a program that will calculate the tip. Ask the user for the value of the bill and how much, in percentage, they want to give as tip, then print the value of the bill, the percentage and the total value, including the tip.
-

Loops

What are loops? Imagine you're drawing a circle. When you reach the end of the drawing, you find the beginning of it. In programming, loops are used in code repetition, when the code block reaches the end, it starts again. Up until now, our code began and ended, from top to bottom. But what if we want the code to run a certain amount of times, or if we want it to run until a condition is true?

Note: Each time a loop starts over it's called an iteration.

For loop

The `for` loop is used when we know how many times the code needs to be repeated. It is used utilized in conjunction with the function `range`. Let's analyze the `for` loop a little more:

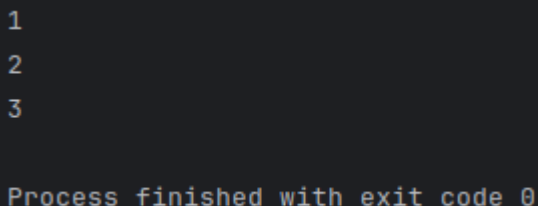
```
for number in range(1, 4):  
    print("Hello")
```

First we use the command `for`, then we create a variable that I've decided to call `number`, we then use the command `in` and call the function `range`. Another way of reading this would be "for each number from 1 to 4, not including 4, do this".

When the loop begins, the variable `number` will have the value of 1, python will run the commands in the block of code and then will return to the beginning of the `for` loop, this time, `number` will have the value of 2, the block of code will run once more and go back to the beginning, and now `number` will have the value of 3, the block of code will run and the loop will end.

We can access the variable `number` inside the `for` loop.

```
for number in range(1, 4):  
    print(number)
```



```
1  
2  
3  
  
Process finished with exit code 0
```

As you can see here, on the first loop, `number` had the value of 1 and that was printed on screen, on the second it had the value of 2 and on the third it had the value of 3.

Nested for loops

Just like with `if`, we can create a `for` loop inside another;

```
for i in range(1, 5):  
    print(i, end=': ')  
  
    for j in range(1, 5):  
        print(j, end=' ')  
  
    print()
```

I know this code is a little scary, so let's read it together. The first loop begins and the variable `i` has the value of 1, the block of code then runs. In it, we print `i`; Then the second `for` loop begins. In it, the variable `j` will be printed 4 times (from 1 to 5, not including 5), the second `for` loop ends and we go back to the block of code of the first loop, and skip a line with the function `print()`.

This is the result:

```
1: 1 2 3 4
2: 1 2 3 4
3: 1 2 3 4
4: 1 2 3 4

Process finished with exit code 0
```

Note: The function `print()` normally breaks the line after printing. We can change this behavior by passing the parameter `end=`. With it we can define if after each print there will be a space, a line break, a comma, etc.

It is important to note that the variable `i` can be accessed anywhere within the block of code of the first `for` loop, even inside the second `for` loop, this is why I've used different names for each of them.

```
for i in range(1, 5):
    print(i)
    # The variable i can be accessed here, but j cannot.

    for j in range(1, 5):
        print(i, j)
        # both the variables i and j can be accessed here.
    print()
```

range()

Although we have seen how the function `range()` works, I want to go over it in a little more detail. The function `range()` can have up to 3 parameters:

```
range(beginning, end, step)
```

The first, which here I'm referring to as `beginning`, defines in which number the count will begin; The second, called `end`, defines in which number the count will end, not including this number; the third, which I'm referring to as `step` will determine how the count will be done, one by one, two by two, three by three. The default is one by one.

```
for num in range(0, 11, 2):
```

```
print(num)
```

```
0
2
4
6
8
10

Process finished with exit code 0
```

Note: If we pass only one value as an argument, python will count from 0 to this value, without including it. `range(3)` will result in 0, 1, 2.

While loop

The `while` loop is executed while a condition is true. It is used when we don't know how many times a code will need to run.

```
number = 1

while number <= 5:
    print(number)
    number += 1
```

In this example, python will test if `number <= 5` is true, in case it is, it will run the block of code, print the value of number and add 1 to it; case it isn't, the loop ends.

```
1
2
3
4
5

Process finished with exit code 0
```

Important: Be careful so that you don't create an infinite loop. If the condition never becomes false, the program will run infinitely and, eventually, crash - it will stop working and close.

truthy and falsy values

When we try to pass data as conditions, instead of an expression, python will evaluate that data as `True` or `False`. When the value is true, we call it "truthy", and when it's false, we call it "falsy". For example:

```
while 10:
    print("truthy")
    # this is an infinite loop
```

The data `0`, `0.0`, and `''`, are falsy. Any other data is truthy.

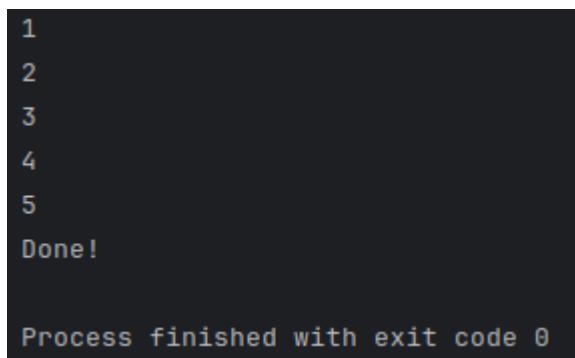
Break

`break` is a command used inside of a loop. It forces python to stop the loop and leave it.

In the following code, the loop will run and print the value of `num` while its value is not bigger than 5, when this happens, python will leave the loop and continue reading the code outside of it.

```
for num in range(1, 10):
    if num > 5:
        break
    print(num)

print("Done!")
```



```
1
2
3
4
5
Done!

Process finished with exit code 0
```

Continue

The command `continue` is also used inside of a loop. When python reaches this command, it passes to the next iteration of the loop.

```
for num in range(1, 11):
    if 2 < num < 8:
        continue
    print(num)

print("Done!")
```

Here, we're telling python that, when the value of `num` is between 2 and 8, to jump to the next iteration of the loop. Which means, when `num` is equal to 3, python will get to the `continue` command, go back to the beginning, and `num` will now have the value of 4, and so on, until `num` is bigger than or equal to 8.

This will be our result:

```
1
2
8
9
10
Done!

Process finished with exit code 0
```

Exercise 4

1. Make a program that will print the number from 1 to 10 and say if they're odd or even.
2. Make a program that will print the numbers from 1 to 50, however, in case the number is divisible by 3, print "Fizz", if its divisible by 5, print "Buzz" and if it's divisible by 3 and 5, print "FizzBuzz". Hint: 15 is supposed to be "FizzBuzz".
3. Make a program that will print the following drawing:

```
#
##
###
####
#####

Process finished with exit code 0
```

Bonus: Flow control in Java

You might have already seen codes with a syntax that is different from python's, with parenthesis, curly braces, with a much scarier look. But the truth is that it's nothing more than that: looks. The concepts are the same in every language. Of course, each language has their own particularities, but the base is the same.

In this bonus section I want to show you how the control flow would be in the java language and explain them to you so that when you find code written in languages other than python, you are able to read them without any issues.

if/else if/else

```
int number = 16;
```

```

if (number <= 10) {
    System.out.println("Small number");
} else if (number <= 30) {
    System.out.println("Medium number");
} else {
    System.out.println("Big number");
}

```

It might be that you can read this code in java without many issues, but let's analyze them together anyways.

First we create a variable of type `int` and assign it the value 16 and then we begin with our `if`. Between parenthesis we have our condition that will be verified and, within curly braces, we have our block of code that will be executed if the condition is true. In this case, the command `System.out.println()`, is the java equivalent of python's `print()`. Another thing that is different is that, in java, instead of `elif`, we use `else if`.

For loop

```

for (int i = 0; i < 10; i++) {
    System.out.println("i = " + i);
}

```

This one is a little scary, huh?

In this type of `for` loop, inside the parenthesis, we create a variable `i` (but it can be called anything, this is just the standard) of value 0, define that the loop will run while `i < 10`, and then we write `i++`, which is the equivalent of `i += 1` in python. After running the code that is between curly braces, 1 will be added to the value of `i`. This code will print on screen the concatenated string `"i = " + i`.

The equivalent of this `for` loop in python is:

```

for i in range(0, 10):
    print(f"i = {i}")

```

While loop

```

int i = 0;

while (i < 10) {
    System.out.println("i = " + i);
    i++;
}

```

The `while` loop has a syntax similar to python's. First we create a variable of type `int` of value 0 to create the condition. The loop will run while `i < 10`, it will print on screen `"i = " + i`, then add 1 to the value of `i` before checking the condition again.

The python equivalent of this `while` loop is:

```
i = 0

while i < 10:
    print(f"i = {i}")
    i += 1
```
