Conditional Statements

You have earned 0 point(s) out of 0 point(s) thus far.

Structure of the 'if' Statements

In some cases, your program needs to execute some part of the code only if a specific condition is true. The simple structure of an if statement is :



if condition:

body # the body code should start at indentation (four spaces)

The **if** statements check the **condition**. The **condition** is always a **Boolean** expression, that is, its value equals either True or False.

If it evaluates to True, it executes the **body** of the **if** statement. If it evaluates to False, it skips the **body**. This logic works like the English language.

input :



if True:

print('it is true')

output :



it is true

input :



empty\_seat = 14

if empty\_seat > 3: # in this case, 14>3=True, so the body will execute

print('there is still seat to sit')

output :



there is still seat to sit

**💡Tips :**

* Note that the condition ends with a colon and a new line starts with an indentation.

### Comparison Operators

Boolean values basically make it clear if a piece of code needs to be executed. Because comparisons result in bool, it's always best to use them as a condition.

Therefore, it is time to examine **comparison operators** :



|  |
| --- |
| *Comparison Operators* |

| **Operator** | **How it works ?** | **Sample** |
| --- | --- | --- |
| **==** | Returns True if two values are equal or False if different | 2 == 2 (True), 2 == 3 (False) |
| **!=** | Returns True if two values are not equal or False if equal | 2 != 2 (False), 2 != 3 (True) |
| **>** | Returns True if the value on the left is greater than the value on the right otherwise returns False | 3 > 2 (True), 2 > 3 (False) |
| **<** | Returns True if the value on the left is less than the value on the right otherwise returns False | 2 < 3 (True), 3 < 2 (False) |
| **>=** | Returns True if the value on the left is greater than or equal to the value on the right otherwise returns False | 3 >= 2 (True), 3 >= 3 (True), 2 >= 3 (False) |
| **<=** | Returns True if the value on the left is less than or equal to the value on the right otherwise returns False | 3 <= 2 (False), 3 <= 3 (True), 2 <= 3 (True) |

**⚠️Avoid ! :**

* Do not use assignment operator **=** in comparison statements. In comparisons, you have to use **==** operator for equality.

Now, let's learn te subject through an example :

input :



x = 6

y = 9

print ("is x equal to y? :" , x == y)

print ("is x not equal to y? :" , x != y)

print ("is x less than y? :" , x < y)

print ("is x greater than y? :" , x > y)

print ("is x less than or equal to y? :" , x <= y)

print ("is x greater than or equal to y? :" , x >= y)

output :



is x equal to y? : False

is x not equal to y? : True

is x less than y?? : True

is x greater than y? : False

is x less than or equal to y? : True

is x greater than or equal to y? : False

'if-else' Statements

In this part of the topic, we will first examine if-else statements.

An **if-else** statement is another kind of conditional statements in Python. It is used with an additional keyword: else.

else works like an if statement. If none of the conditions in if are ensured, "else" will be used to specify all remaining conditions. The simple structure of an if-else looks like :



if condition1:

execute body1

else : # if condition1 is not ensured execute body2

execute body2

**💡Tips :**

* Note that, else doesn't require any condition and the body2 requires 4-space indentation here.

With a few simple examples, you can be sure you will understand this topic much better.

input :



course = 'clarusway'

if course == "clarusway":

print("you guaranteed the job")

else:

print("think about it again")

output :



you guaranteed the job

https://drive.google.com/uc?export=view&id=1AAtBbH3F0uSt3rVDf7PhBS7L1VcMQAh-**Scratch Time ! :**Solve this example with [**scratch**](https://scratch.mit.edu/projects/350671406/editor/).

input :



number = 5

if number <= 3:

print("Number is smaller than or equal to 3")

else: # Optional clause (you can only have one else)

print("Number is bigger than 3")

output :



Number is bigger than 3

'if-elif-else' Statements

The elif statement is used when it requires to specify several conditions in our program.

In Python you can deﬁne a series of conditionals using :

* if for the **ﬁrst** one,
* elif for the **rest**, up until the ﬁnal (optional),
* else for **anything not caught by the other conditionals**.

The basic structure of these statements looks like :



if condition1:

execute body1

elif condition2:

execute body2

else:

execute body3

How does the structure of statement work? : condition1 is checked first, and if the result is True the body1 in the 'if' statement' will be executed. If not, condition2 in the 'elif' statement is checked. If the result is True, body2 will be executed, if not, body3 in the last control point 'else' will be executed.

Let's take a look at the example below:

input :



weight = 80

if weight > 100:

print("That's too heavy!")

elif weight > 75:

print("I can lift that!")

else:

print("That's too light!")

output :



I can lift that!

https://drive.google.com/uc?export=view&id=1AAtBbH3F0uSt3rVDf7PhBS7L1VcMQAh-**Scratch Time ! :**Solve this example with [**scratch**](https://scratch.mit.edu/projects/340876151/editor/).

We can use as many elif statements as we need, so your conditions can be varied.

| **if statements** |
| --- |
| *Diagram of General 'if-elif-else' Statement* |

As we stated before, the code inside the else block is executed only if all conditions before it is False. Let's see it in an example :

input :



audience = "baby"

if audience == "kid":

print("it is free to go to cinema")

elif audience == "teen":

print("discounted price!")

elif audience == "adult":

print("normal price")

else:

print("No such audience, stay at your home!")

output :



No such audience, stay at your home!

https://drive.google.com/uc?export=view&id=1AAtBbH3F0uSt3rVDf7PhBS7L1VcMQAh-**Scratch Time ! :**Solve this example with [**scratch**](https://scratch.mit.edu/projects/341440492/editor/).

In this program, we grouped the prices of movie tickets based on the age ranges of the audience going to the cinema. If you pay attention, after the True or False check-in each level, the flow continues according to the response received. In the last step, else can be reached if all the answers are False. See the flow diagram below :

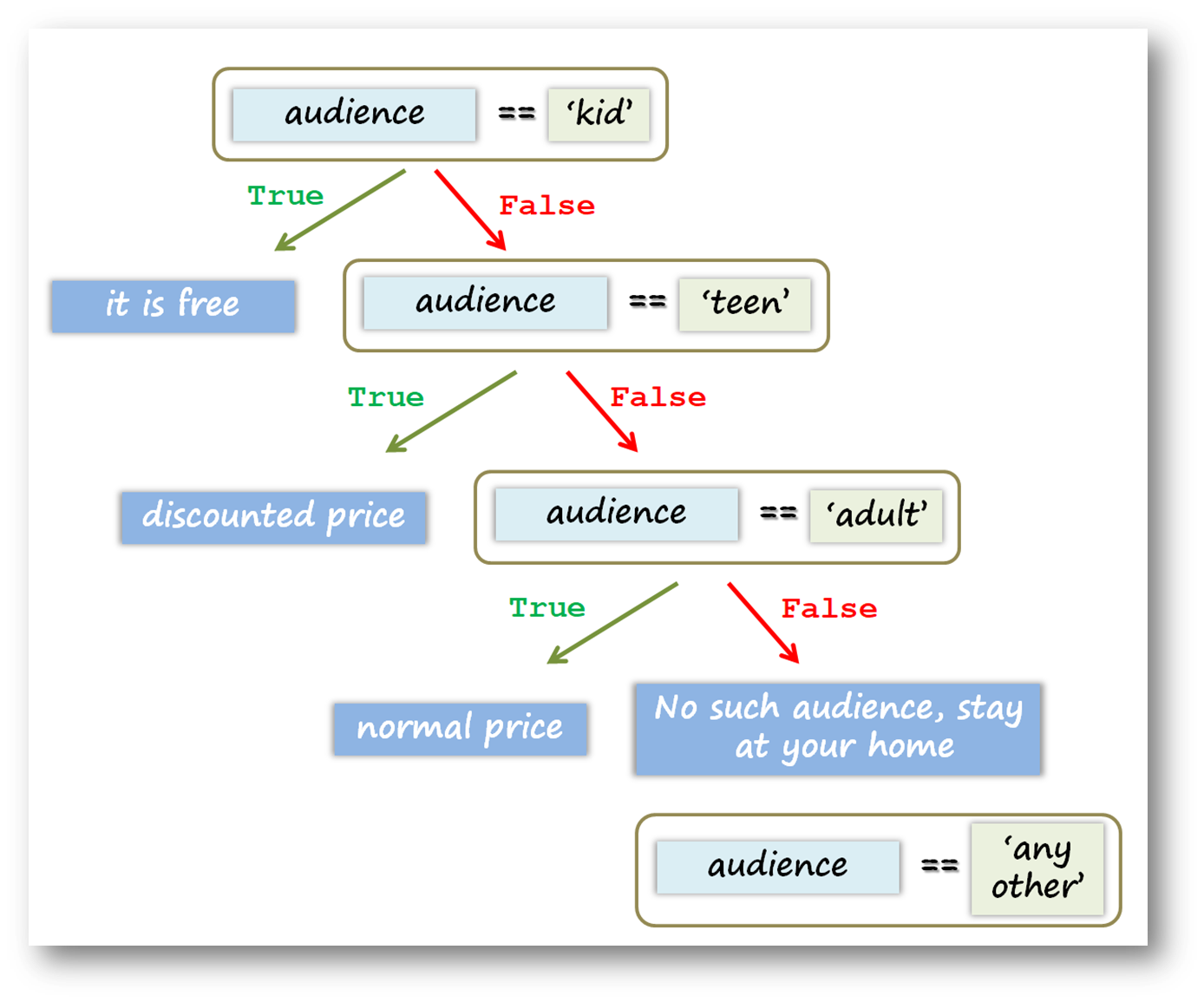


Diagram of Logic Flow

Nested 'if-elif-else' Statements

Both if-else and if-elif statements can be nested. Let's see the nested structure on the same movie ticket example.

input :



audience\_group = 'kid', 'teen', 'adult'

audience = "teen"

if audience in audience\_group:

if audience == "kid":

print("it is free to go to cinema")

elif audience == "teen":

print("discounted price!")

else: # audience == "adult":

print("normal price")

else:

print("No such audience, stay at your home!")

output :



discounted price!

Let us write a program that asks you to enter your exam score and calculates the range in which your degree is based on your exam score. Try to run this code on your *Jupyter Lab* cell if available.



score = int (input("Enter your score :"))

if score >= 90:

if score >= 95:

Score\_letter="A+"

else:

Score\_letter="A"

elif score >= 80:

if score >= 85:

Score\_letter="B+"

else:

Score\_letter="B"

else:

Score\_letter="below B"

print ("Your degree: %s" % Score\_letter)

**💡Tips :**

* input() is a function that takes a value from the user and assigns it to a variable that you choose. We will use it very common.

Nested if-elif-else structures may seem a bit complicated to you, the best way to overcome this is to examine and practice plenty of samples.

If you eager to find **more on control flow tools** see [**here**](https://docs.python.org/3.8/tutorial/controlflow.html?highlight=else#more-control-flow-tools).

Loops

Definitions

When writing programs in Python, in some cases it is not enough to execute our block of code only once. The **loops** are used to repeat (iterate) the execution of a block of code.

As one of the most main functions in programming, loops are an important part of almost every programming language. Loops enable programmers to set certain sections of their code to repeat through a number of loops which are referred to as iterations.

This topic covers using multiple types of loops and applications of loops in Python. You will learn two types of loops which are :

* **while Loop**,
* **for Loop**.

Q: What are the two major loop statements?  
A: for and while loops.

### 'while' Loop

while loops have a boolean logic, similar to if statements. As long as the result of the condition returns True, the code block under while loop runs. When the condition returns to False, the loop execution is terminated and the program control moves further to the next operation. Here is the simple structure of a while loop :



while condition:

body

| **while_loop** |
| --- |
| Diagram of Basic Structure of *while* Loop |

We will not use this loop as often as the for loop, but it is still worth understanding.

Lets create our first while loop :

input :



number = 0

while number < 6:

print(number)

number += 1

print('now, number is bigger or equal to 6')

output :



0

1

2

3

4

5

now, number is bigger or equal to 6

The variable number acts as a counter in this loop. This variable changes its value after each iteration. When the value of a counter reaches 6, the program control moves to the next operation and prints the text above.

**⚠️Avoid ! :**

* If we make a logical mistake in the loop variable (since you don’t increase your variable, a condition never becomes False and can work forever), we can start an infinite loop! For this reason, we have to specify a condition for the loop to give False to exit the loop.

We can call a list in while loop. Let's see an example :

input :



my\_list=["a", "b", "c", "d", "e"]

a = 0

while a < len(my\_list):

print('square of {} is : {}'.format(a, a\*\*2))

a+=1

output :



square of 0 is : 0

square of 1 is : 1

square of 2 is : 4

square of 3 is : 9

square of 4 is : 16

https://drive.google.com/uc?export=view&id=1AAtBbH3F0uSt3rVDf7PhBS7L1VcMQAh-**Scratch Time ! :**Solve this example with [**scratch**](https://scratch.mit.edu/projects/341441228/editor/).

**💡Tips :**

* Did you remember? variable += number is the same as variable = variable + number.

Always use valid syntax and make comments. In the beginning, it may seem that the while loop is not so easy to apply, but several times later, you’ll understand that it’s a very useful tool. Lastly, let's play famous 'guessing a number game' using while loop :



answer = 44

question = 'What number am I thinking of? '

print ("Let's play the guessing game!")

while True:

guess = int(input(question))

if guess < answer:

print('Little higher')

elif guess > answer:

print('Little lower')

else: # guess == answer

print('Are you a MINDREADER!!!')

break

In the example above;

* We have written a program that does not exit the while loop until you find the correct number,
* We used break keyword in order to quit and exit the while loop,
* When the user knows the answer (44) and enters input, it takes the value of 44 and assigns to variable guess, in the end, else works and breaks the loop.

☝ Discuss it in-class!

If available, run this code on your VS Code with JupyterNotebook cell and try to understand how it works. Enjoy!

### 'for' Loop

You'll learn one of the most used, very simple and useful syntaxes in Python: for loop. When you want to iterate a block of code you will use for loop. To create a for loop, you need a **variable** and an **iterable object**. Here is the simple structure of a for loop :



for variable in iterable :

code block

Let's examine the subject through an example. We need an iterable so we can use a list.

input :



for i in [1, 2, 3, 4, 5] :

print(i)

output :



1

2

3

4

5

You can follow the iterating steps of for-loop below :

| **Tiobe_Index** |
| --- |
| How 'for-Loop' Works? |

In the structure of the for loop, you can use also an iterable variable of course. See the example below :

input :



seasons = ['spring', 'summer', 'autumn', 'winter']

for season in seasons :

print(season)

output :



spring

summer

autumn

winter

**✏️Homework:**

* for i in {'n1' : 'one', 'n2' : 'two'} : print(i) Will this line of loop work? If **yes** what will be the output? If **no**, what is the problem? (Note: Try to guess the result or do research about this code, before running on the Playground or on your Jupyter Lab.

Q: How does for loop and while loop differ in Python and when do you choose to use them?  
A: **For loop** is generally used to iterate through the elements of various collection types such as list, tuple, set and dictionary.  
  
**While loop** is the actual looping feature that is used in any other programming language. This is how Python differs in handling loops from the other programming languages.

### Working with the Iterators

Let us explain the term **iteration** a little more.

**Iterable** object can be anything for which items are received one by one, forward only. In Python, the process of recurrent execution of a block of code is called an **iteration**.

We can basically classify iterations as two headings :

* If the number of repetitions is predetermined, it is called **definite** iteration.
* The repetition structure that makes the code block run as long as the predetermined condition generates True is called **indefinite** iteration.

| **Loop** |
| --- |
| Diagram of the Iterating Loop |

For example; string, list, tuple, dictionary or set are the iterable types of data.

Let's use a string variable as an iterator in for loop :

input :



course = 'clarusway'

for i in course :

print(i)

output :



c

l

a

r

u

s

w

a

y

https://drive.google.com/uc?export=view&id=1AAtBbH3F0uSt3rVDf7PhBS7L1VcMQAh-**Scratch Time ! :**Solve this example with [**scratch**](https://scratch.mit.edu/projects/341441774/editor/).

**Q**: What are Python iterators?  
**A**: Iterators in Python are array-like objects which allow moving on the next element. We use them in traversing a loop, for example, in a for loop.

### Operations with the 'for' Loop

In this topic, you will learn about how we use the for loop using several functions and methods.

In the example below, you'll get a number from the user and print a sentence the number of times we receive from the user.



times = int(input("How many times should I say 'I love you'"))

for i in range(times):

print('I love you')

As we stated before, input() function can get the value of different data types and assign a variable you chose. In the example above, it gets a number and assigns it to times. If the user enters 3 then the output will be :



I love you

I love you

I love you

Now, let's write a code that asks the user a number between 1 and 10 and puts that number into the **multiplication table**.



n = int(input('enter a number between 1-10'))

for i in range(11):

print('{}x{} = '.format(n, i), n\*i)

☝ Discuss it in-class!

**✏️Homework:**

* Write the same code above by yourself but using 👉🏻**%** operator in print() function.

**💡Tips :**

* If you want the user to input numbers, use the input() function together with the int() function. Otherwise, the value entered by the user will be in the **string** data type.

Let's get to know the features of range() function in details:

The range() function creates an iterable sequence of numbers. And it can be simply converted into an iterable object: list, set, and tuple. For example :

input :



b = list(range(11))

print(b)

output :



[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

input :



a = set(range(0,10))

print(a)

output :



{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

input :



c = tuple(range(11))

print(c)

output :



(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

You can keep in mind the formula syntax below for range() function:

**The formula syntax is : range(start, stop, step)**

Besides, you can use starred expression 👉🏻**\*** before range() function to separate its elements. See these examples :

input :



print(range(5)) # it will not print the numbers in sequence

print(\*range(5)) # '\*' separates its elements

output :



range(0, 5)

0 1 2 3 4

input :



print(\*range(5,25,2))

output :



1

2

5 7 9 11 13 15 17 19 21 23

Starred expression 👉🏻**\*** can separate other **iterable** objects. For example, you can separate a string:

input :



print(\*('separate'))

output :



s e p a r a t e

☝ Discuss it in-class!

You can create reverse sequence numbers using a negative step.input :



print(\*range(10,0,-2))

output :



10 8 6 4 2

In some cases, you will need to set up the for loop with multiple variables and the iterables. Examine the example :input :



text = ['one','two','three','four','five']

numbers = [1, 2, 3, 4, 5]

for x, y in zip(text, numbers):

print(x, ':', y)

output :



one : 1

two : 2

three : 3

four : 4

five : 5

☝ Discuss it in-class!

**💡Tips :**

* zip() function make an iterator that aggregates elements from each of the iterables.

Nested 'for' Loop

As a programmer, you may sometimes need to interact with a single element of an iterable data and all other elements simultaneously, that is, your code block in a loop can also contain a loop. Yes, we're talking about nested loops.

In Python, you can easily place one loop inside another one. First outer loop then inner one runs. You'll see it in the following example :

input :



who = ['I am ', 'You are ']

mood = ['happy', 'confident']

for i in who:

for ii in mood:

print(i + ii)

output :



I am happy

I am confident

You are happy

You are confident

You can better understand how it works, by watching the following animation.

| **Nested_For_Loop** |
| --- |
| *How 'nested-for-Loop' Works?* |

https://drive.google.com/uc?export=view&id=1AAtBbH3F0uSt3rVDf7PhBS7L1VcMQAh-**Scratch Time ! :**Solve this example with [**scratch**](https://scratch.mit.edu/projects/341442714/editor/).

If you want to get deeper into it, you can find the details about the **loops**[**here**](https://docs.python.org/3.8/tutorial/controlflow.html?highlight=else#break-and-continue-statements-and-else-clauses-on-loops).