\*Before you write a program, you should try writing an algorithm which all the orders you want to give.

VARIABLE AND IDENTIFIERS

Identifiers can only include words, numbers and underscores

They cannot start with a number

They cannot have a space and they cannot be a Python keyword

To assign a string to a variable it has to be between “”

If the variable has two (or more) words, join them using underscores

Give representative names to the variables/identifiers

To comment use the symbol #

Variable types

Integer (int)

Float

To know which is the type of the variable you can type “print (type (x))”, with x being the variable.

String (str)

Boolean (bool)

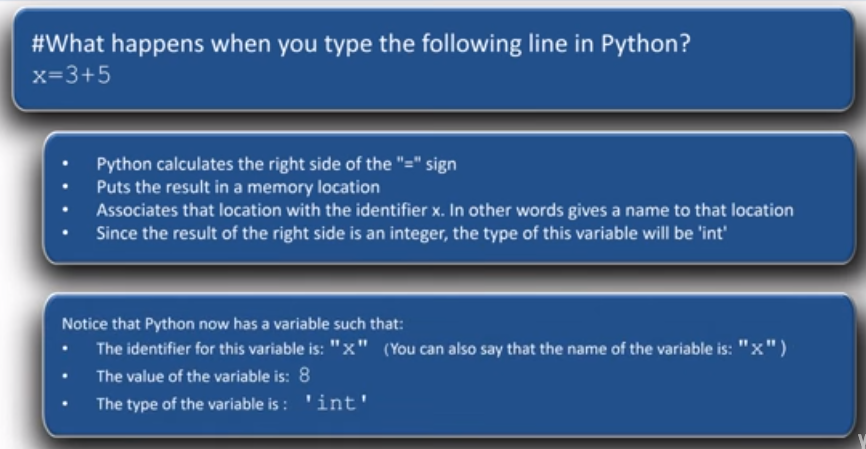
Lists (list)

Dictionaries (dict)

Tuples

…

Integers



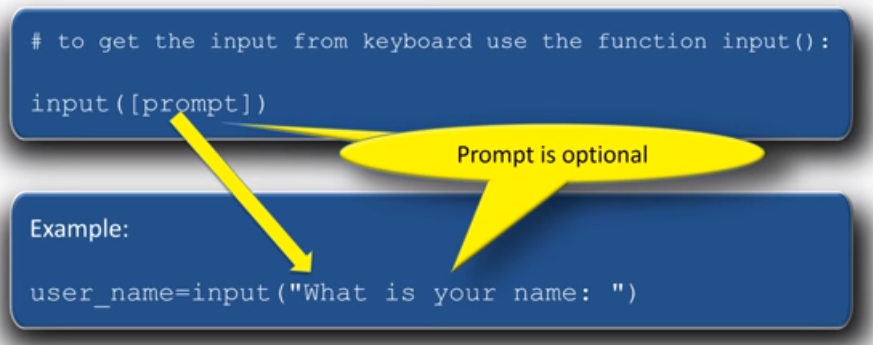
Caution:

Operations with floats result in a float, even if the result is an integer (6.2-2.2)

Divisions always return a float (8/2)

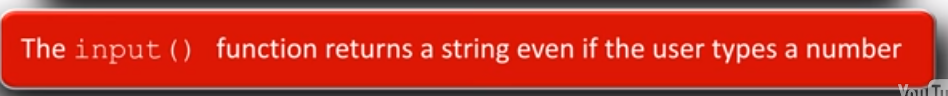


Get input from user

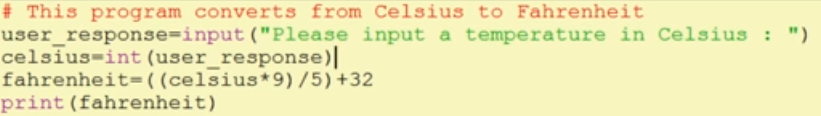


user\_response=input (“What is your name?”)

Print (“Glad to meet you”, user\_response)



If you are asking for a number or other types, then you must convert string to the proper type.

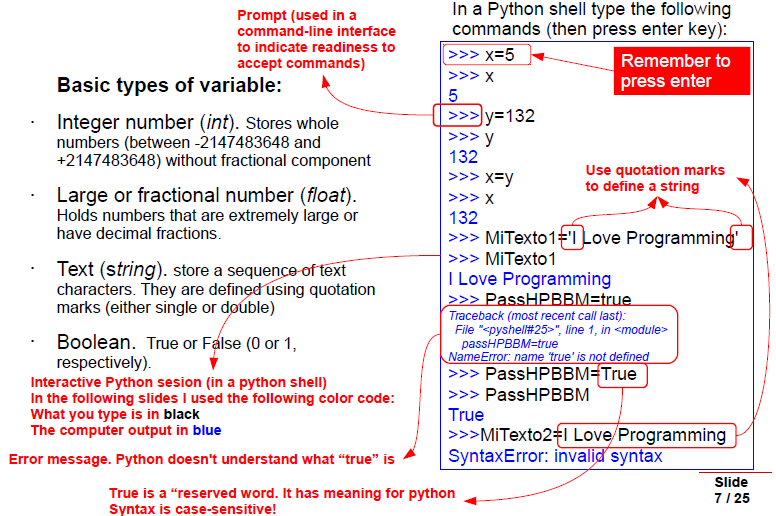


The third line is used to convert a string to an integer so that we can get the result of the operation. If we don’t do it, we will get an error, as Python cannot divide a string with a number.

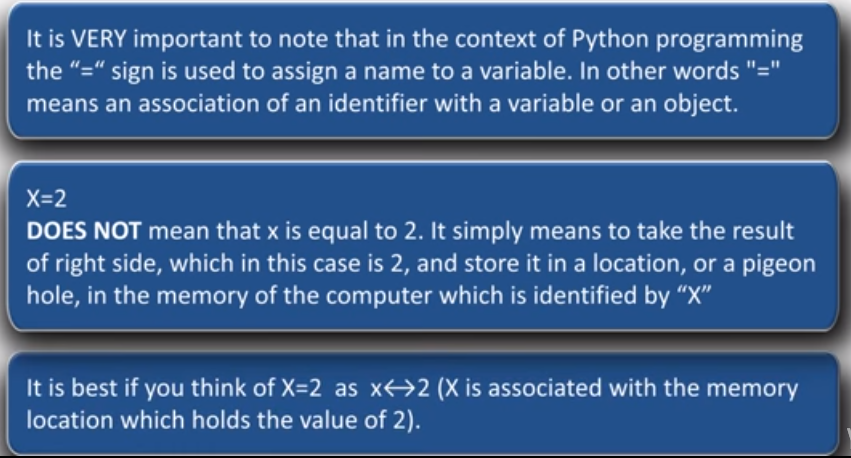
You could also use: user\_response=int(input(“Please input a temperature in Celsius: “))

A similar conversion is used with floats, Booleans…

\*You have to use different names of variables for the string and the number (user\_response and Celsius in this case).



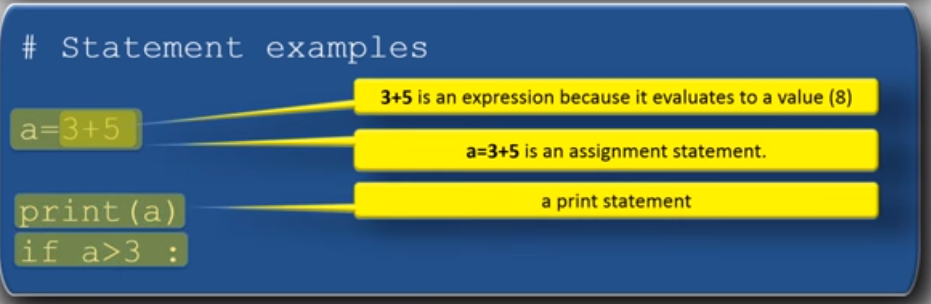
STATEMENTS



Expression: any section of the code that evaluates to a value



Statement: an instruction that Python can execute



Assignment: is a statement that associates names with values in your program.



LIST BASICS

A list is a container that contains a number of other objects.

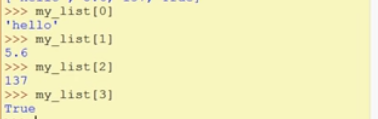
The order of the objects in the list matters. This means that a list is a sequence of objects and not a bag of objects.

To crate a list place all the items (elements) inside a pair of square brackets and separate them by commas

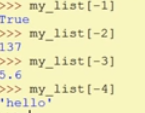
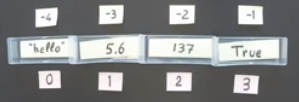
My\_list=[“hello”, 5.6, 135, True] 🡪 The elements can be of different types

There are two ways for accessing list elements:

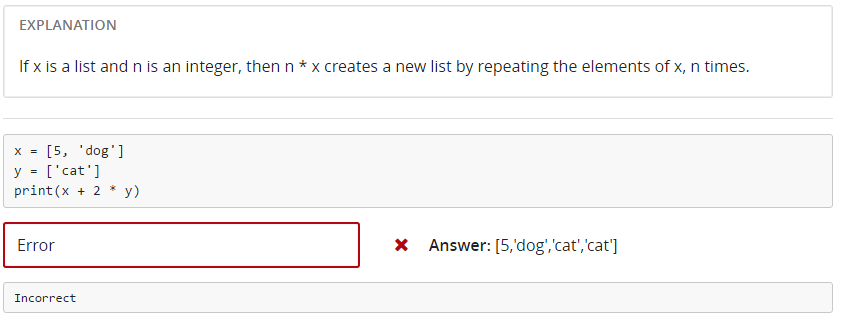
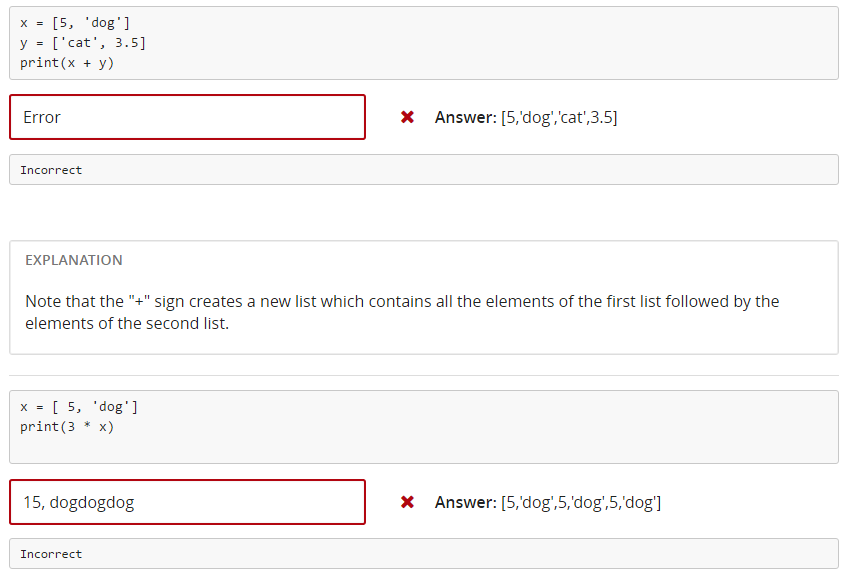
-Positive indexing: each element has an index which increases from left to rught. The first element on the left has index 0, the next element has index 1 and so on.



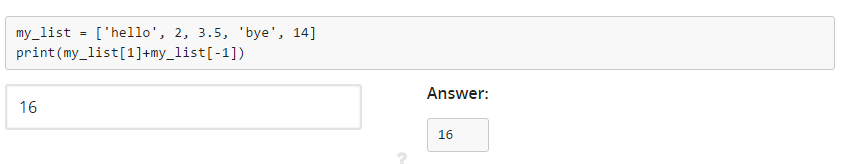
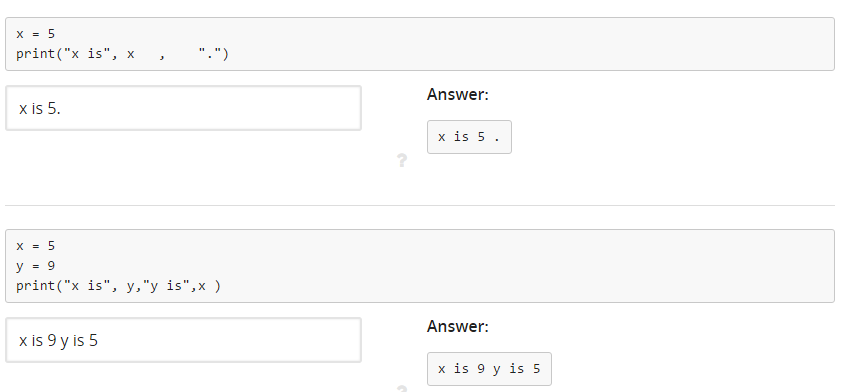
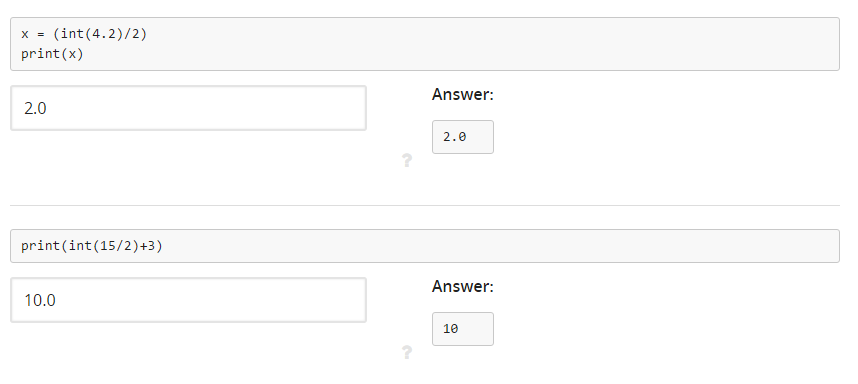
-Negative indexing: goes from right to left. The last element on the right has the index -1, the element before last has index -2 and so on.

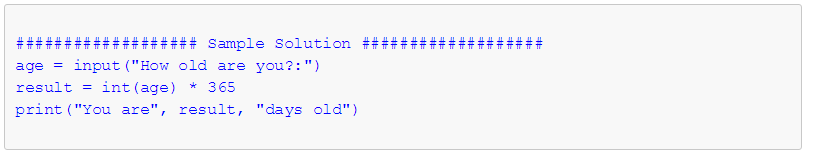


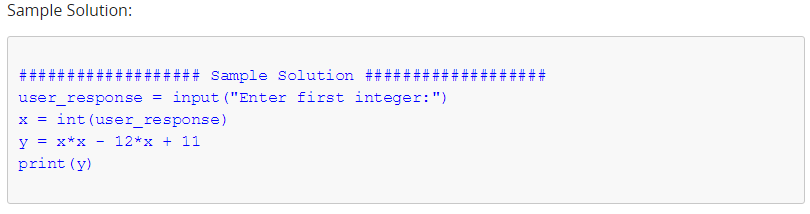
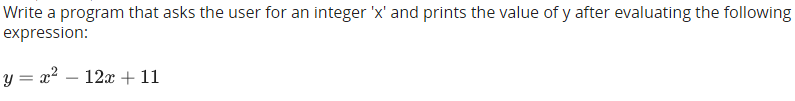
Caution:

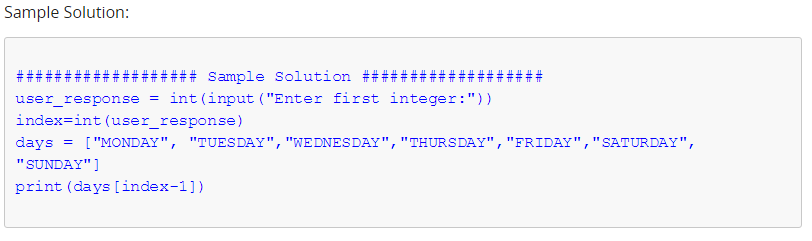
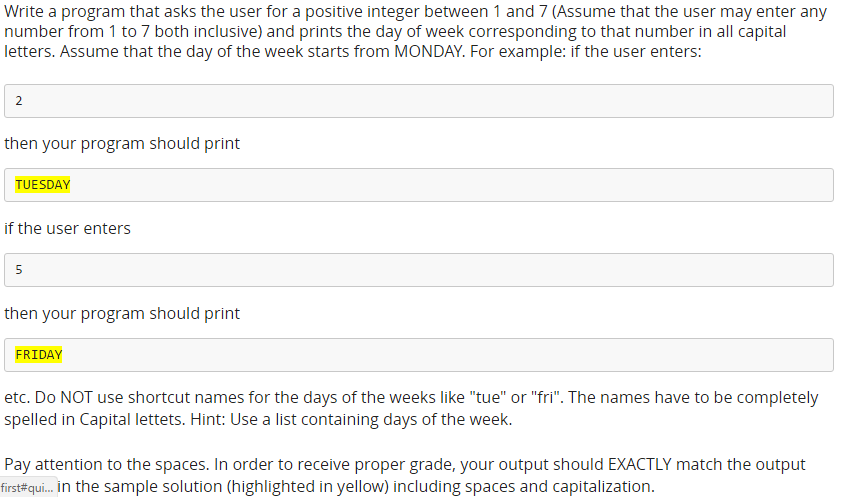


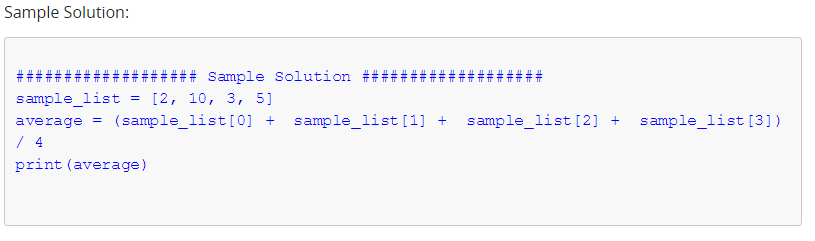
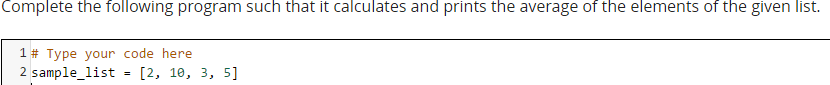
EXAM 1











OPERATORS

Operators are the symbols which tell the Python interpreter to do some mathematical or logical operations.

-Binary operators: it works on two operands (3\*2)

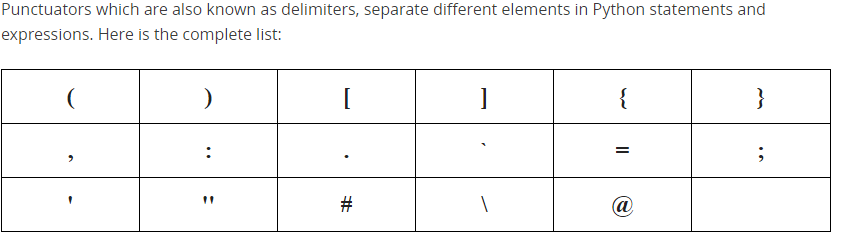
-Unary operators: it works on one operand (. upper)

!= means not equal





Punctuators and delimiters



Specific operators



>>>10.0+15

25.0

>>>15%4

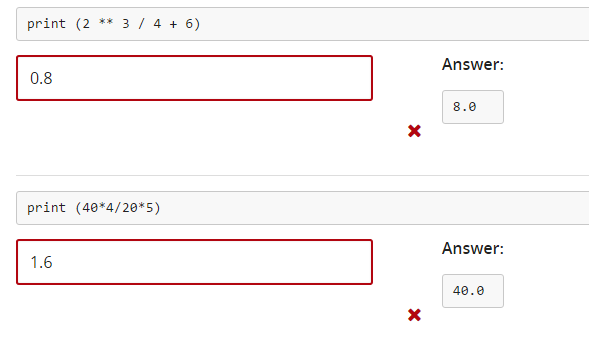
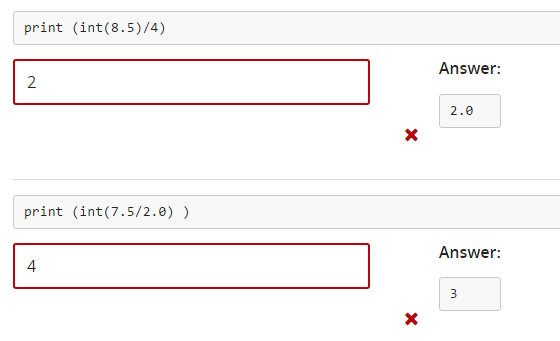
3

>>9//2

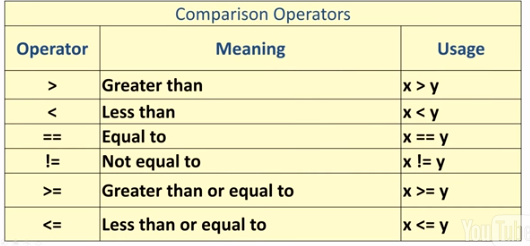
4

>>9.0//2

4.0



Relational operators(comparison operators)



They all return a Boolean

Logical operators



>>> (5>4) and (3>7)

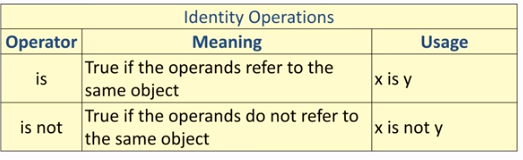
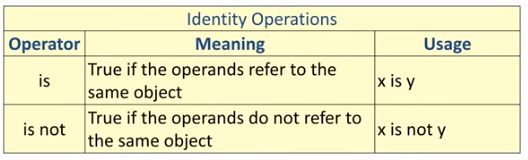
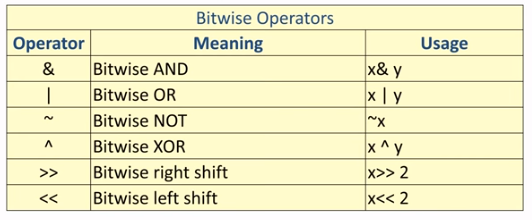
False

>>> (5>4) or (3>7)

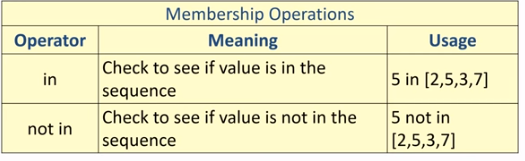
True

>>> not (5>4)

False



Membership operators



>>>x= [6, “cat”, “dog”, 4.5]

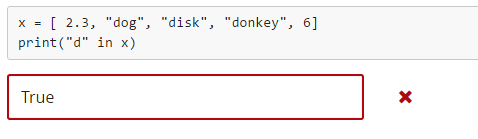
>>>6 in x

True

>>>s=” hello there”

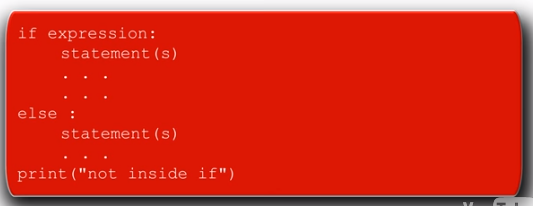
>>>“t” in s

True



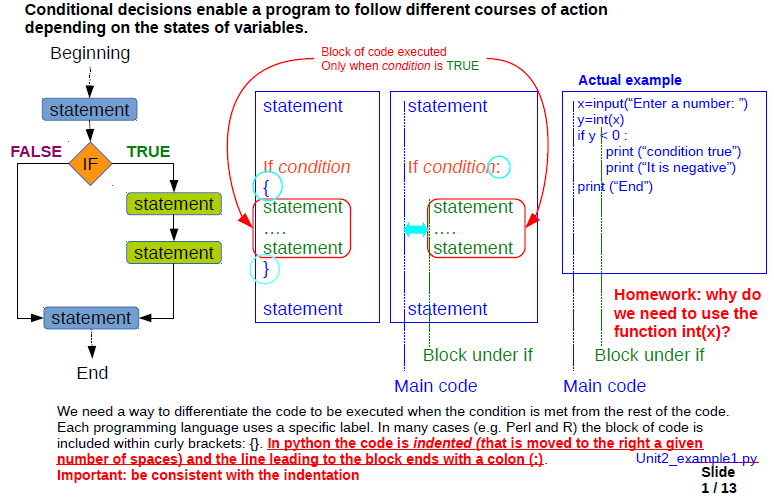
CONDITIONALS (if, else)

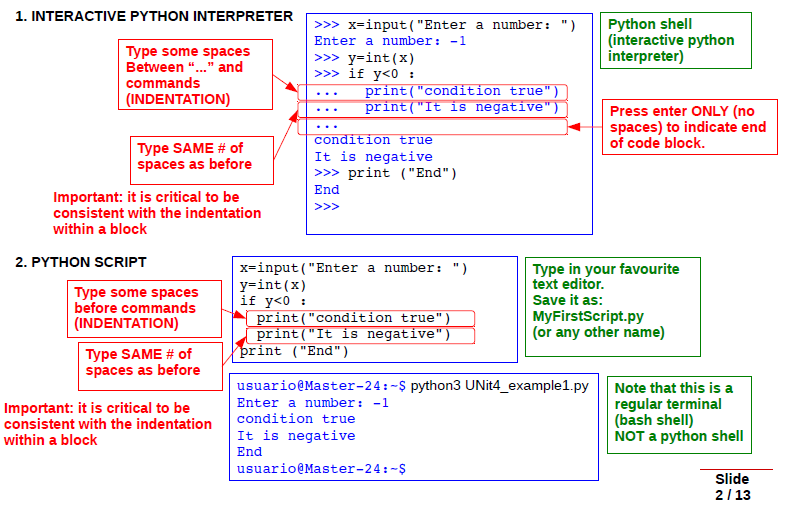
Conditional statements are used to perform different computations or actions deepening on whether a condition is True or False.

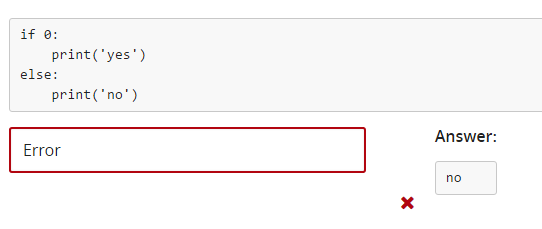


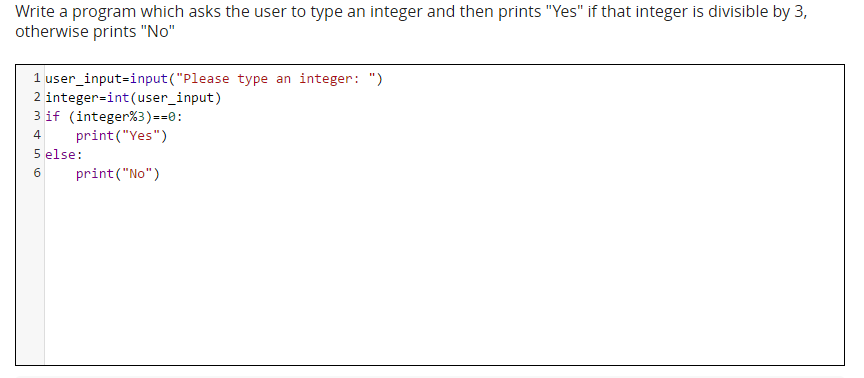
Caution: it is required to put “:” (colon) after the if/else expression.

Tab is always required to begin the conditional statement









If/elif/else statements

If else statements are used when a condition is True or False

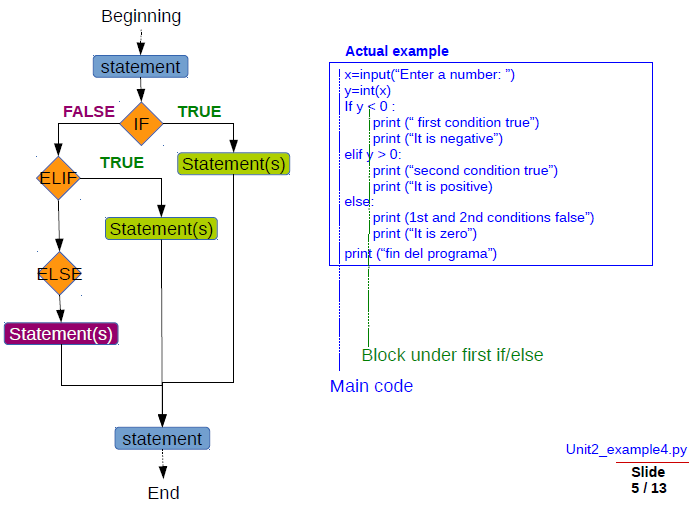
In cases where multiple conditions should be checked you should use:

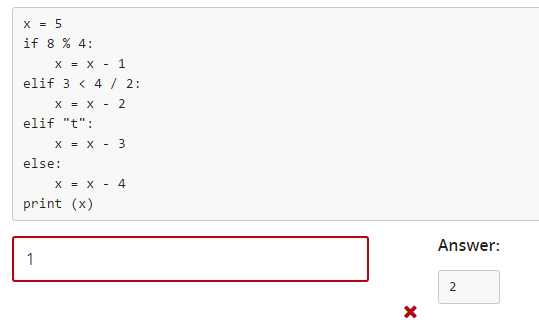
If

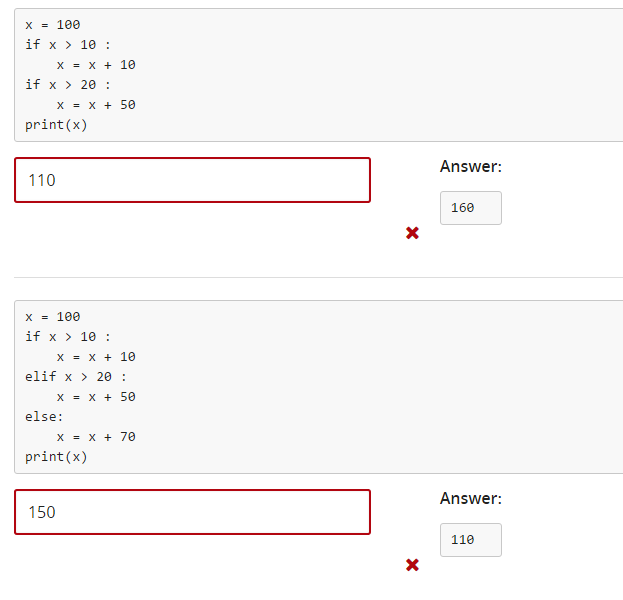
Elif

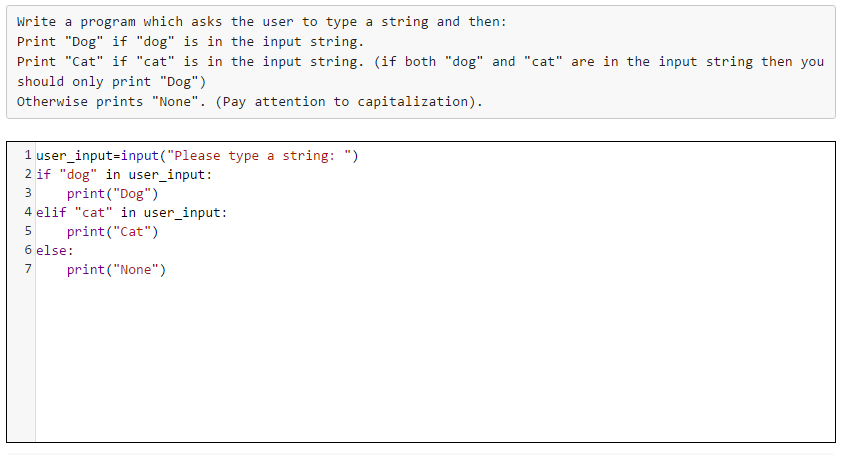
Else

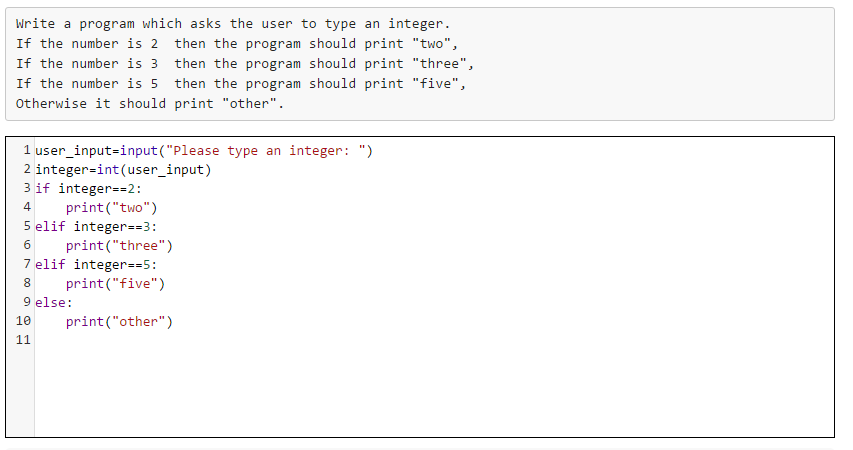




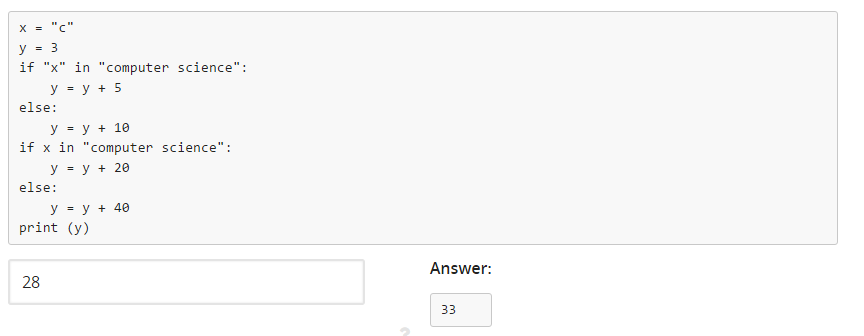
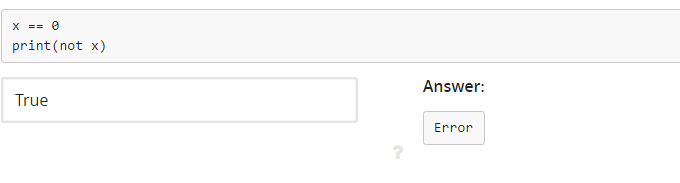


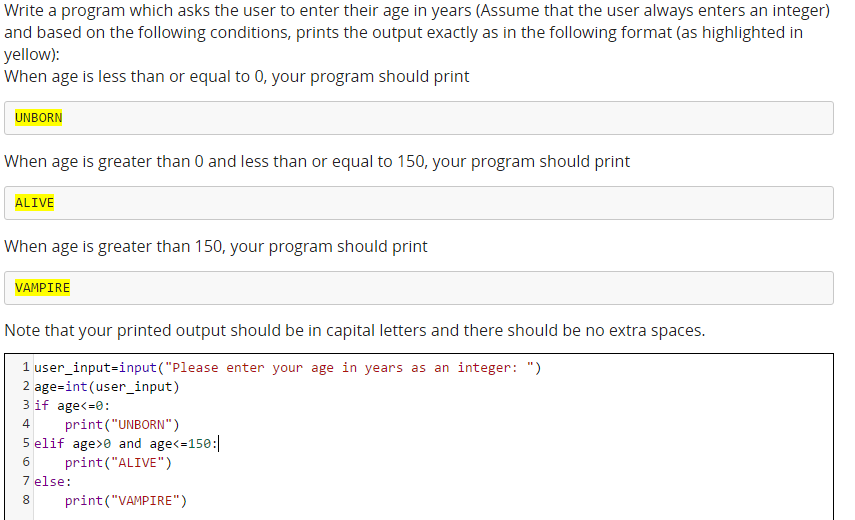


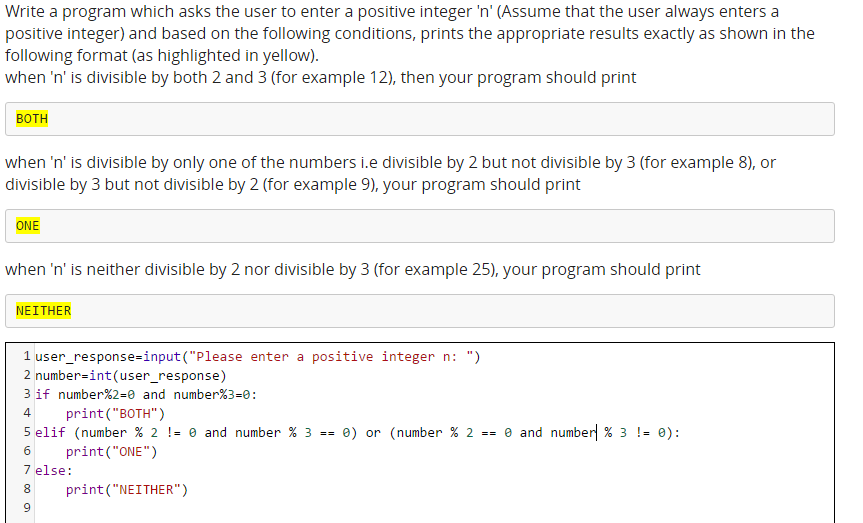


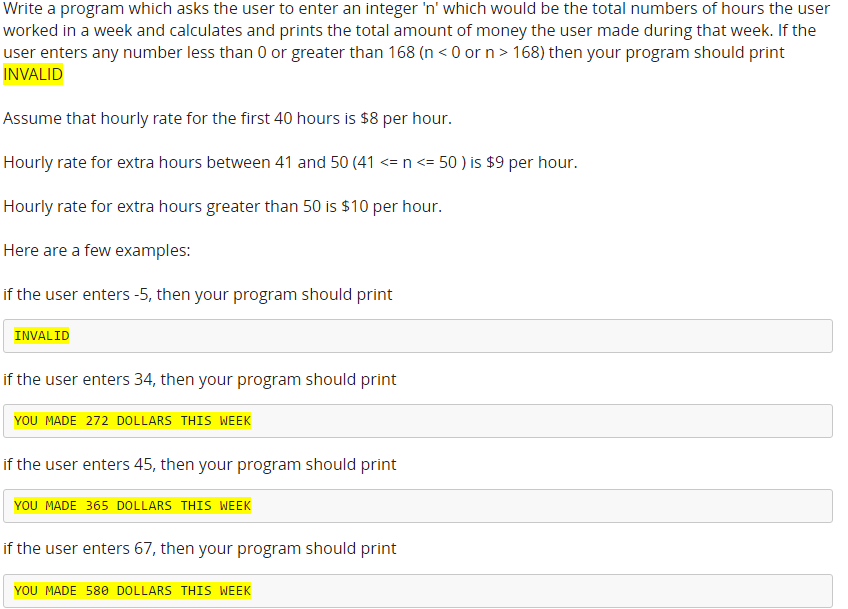


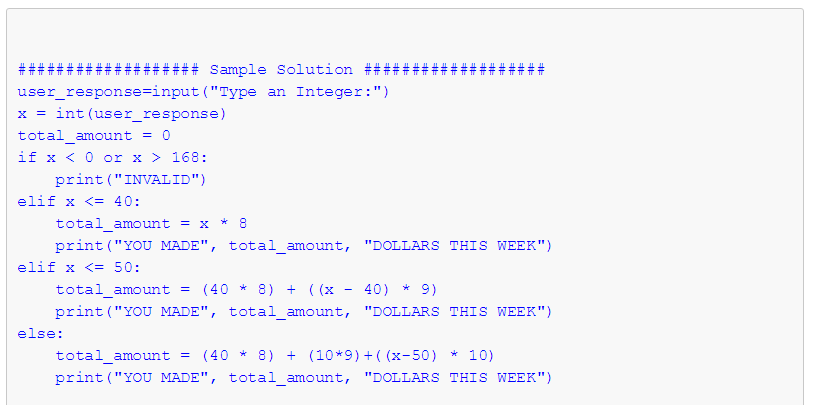
EXAM 2

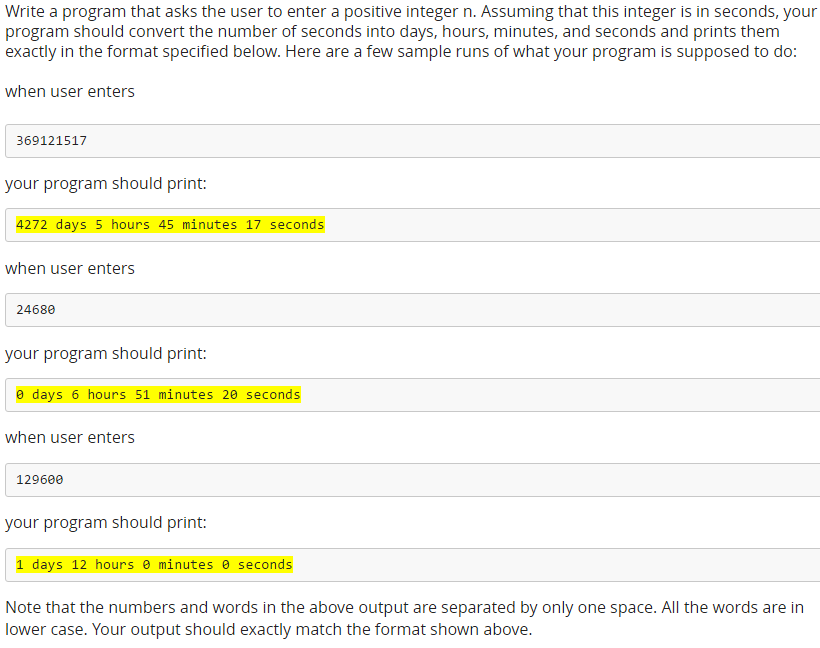


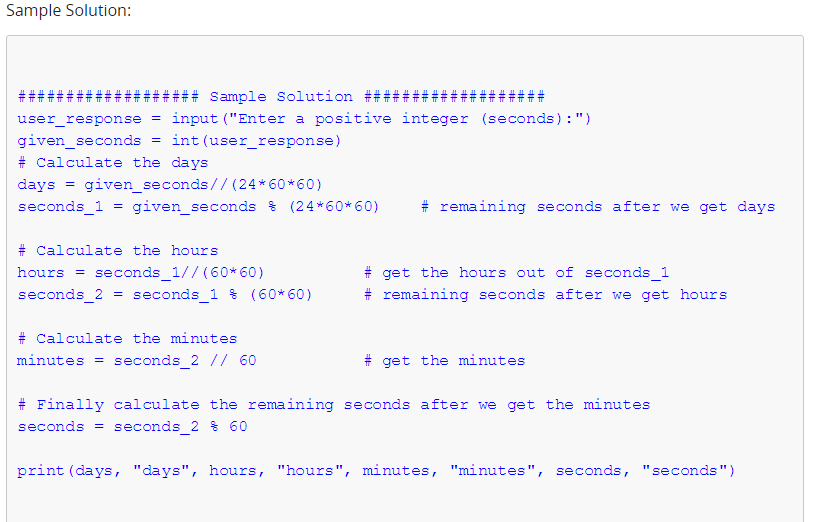












LOOPS

Loops repeat a block of code several times. There are two main types:

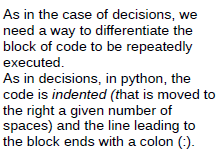
● “***for***” loops. - The block is repeated ***for*** a predetermined number of times

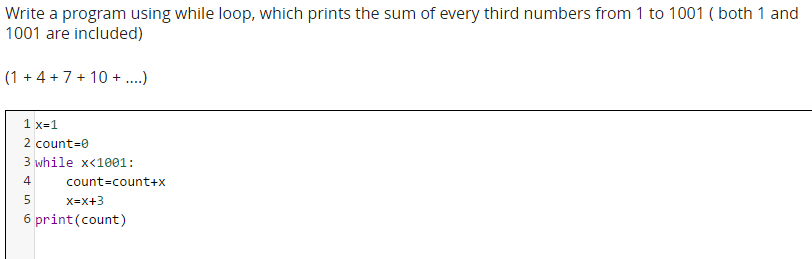
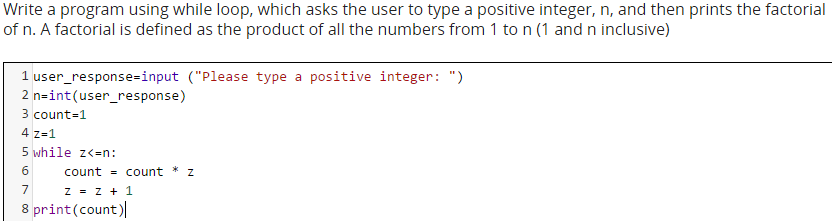
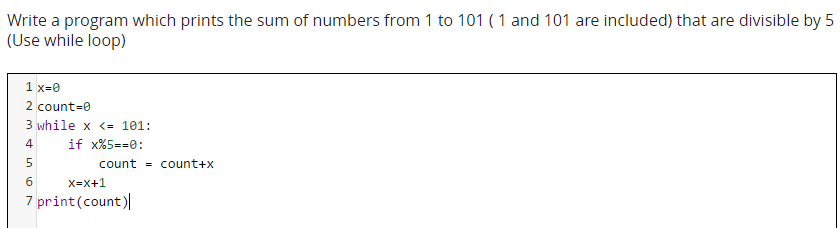
● “***while***” loops. - The block is repeated ***while*** a condition is true.

While Loops

A while loop in Python repeatedly executes one or more statements as long as a condition is True.

If the expression in front of the while statement never comes False, you will have an infinite loop and your program loop endlessly and never exits.

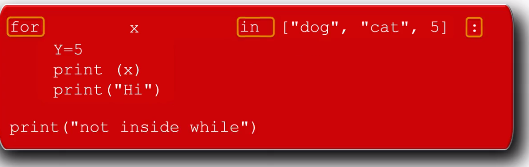


For loops

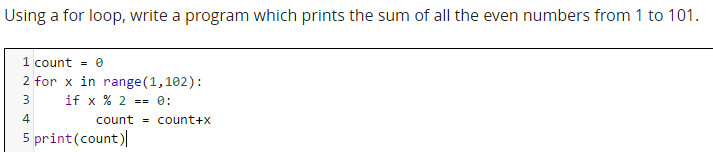
Are used for two different purposes.

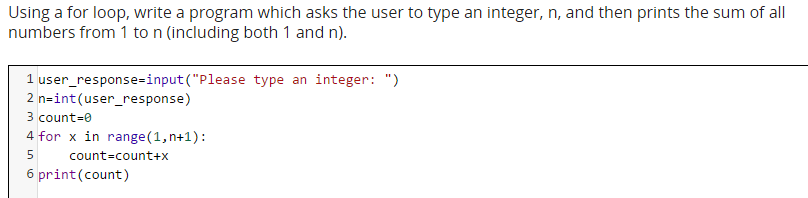
When you want to go over the items of any sequence, such a list or a string

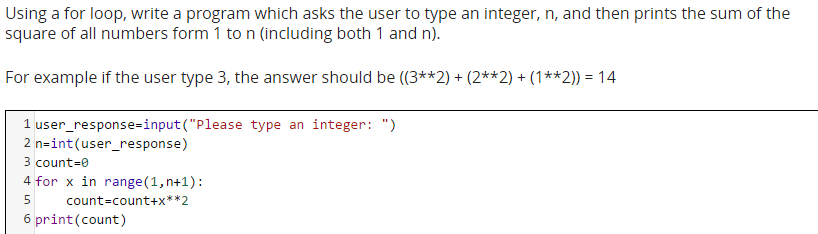
When you want to repeat some piece of code n number of times

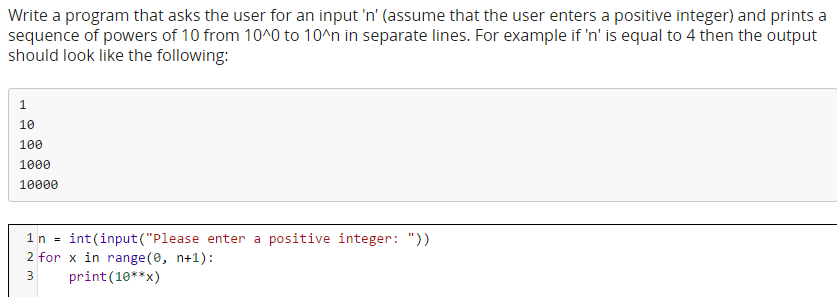


Function range: this function can create a list o numbers automatically. All you have to do is specify the starting number and the end number. For example, if I want to create a list off numbers from 1 to 10, then I will say range(1,11). The range function starts with the start number but it always goes to the last number minus 1.







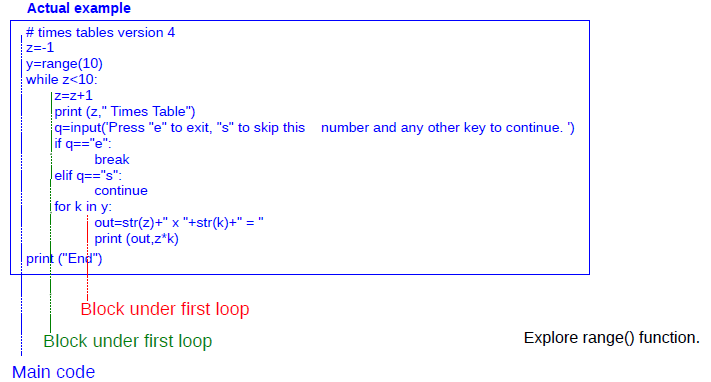


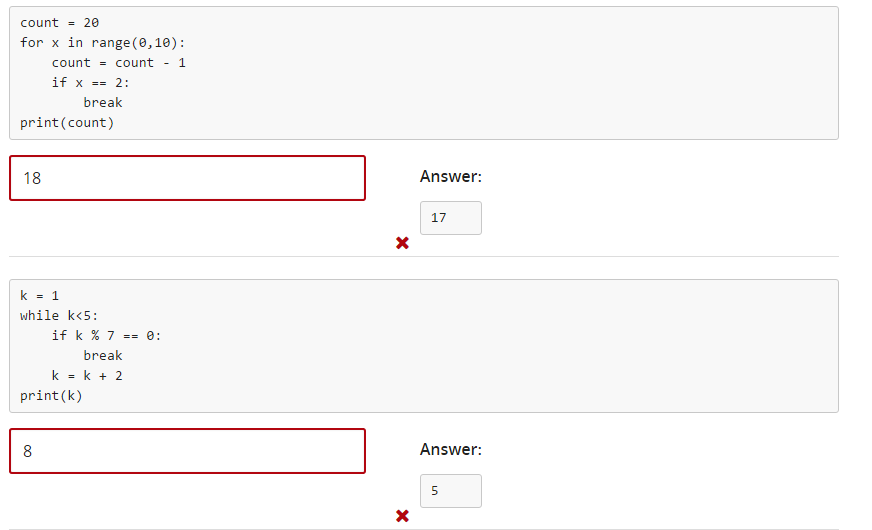
Continue and break

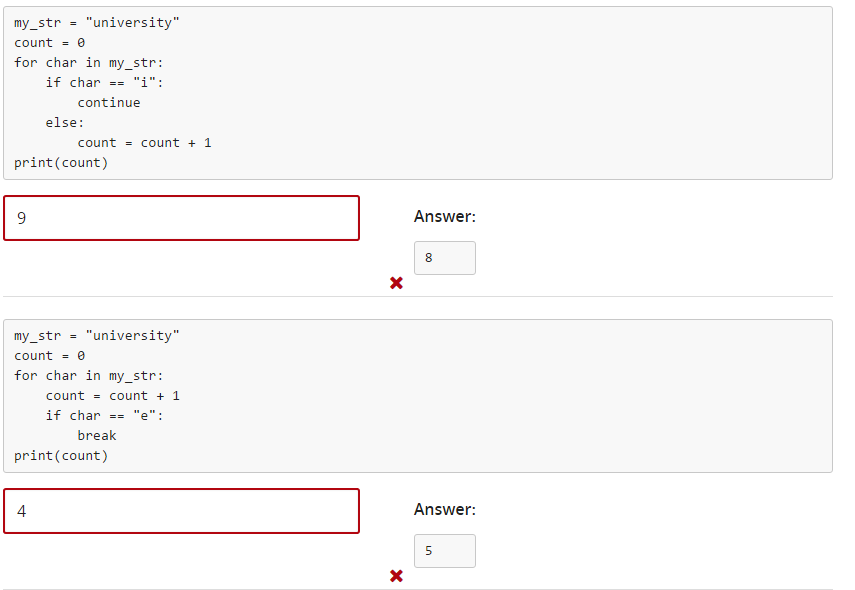
Sometimes it is necessary to terminate the current iteration or terminate the current loop in a for or while loop. continue and break statements are used in these situations.

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

The break statement terminates the loop containing it. The program jumps to the statement immediately after the body of the loop.







NESTED LOOPS

Nested loop is when a loop is inside another loop.

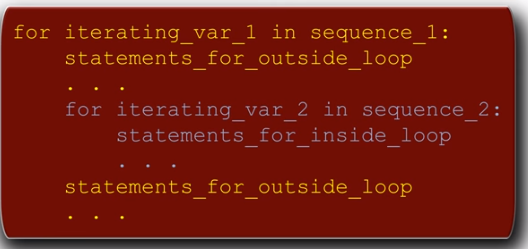
Any type of loop can be nested inside another type. This means:

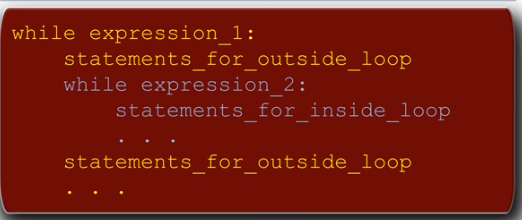
-A for loop can be nested inside another for loop.

-A for loop can be nested inside a while loop.

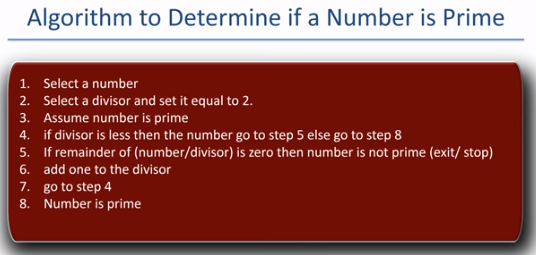
-A while loop can be nested inside another while loop

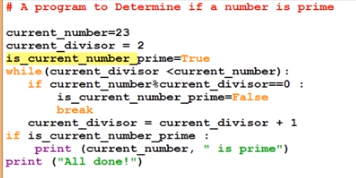
-A while loop can be nested inside a for loop

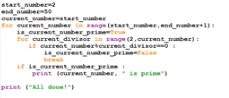
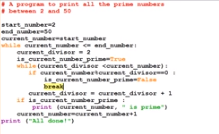


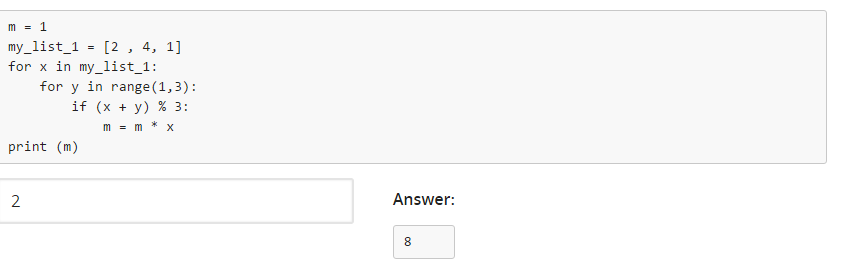
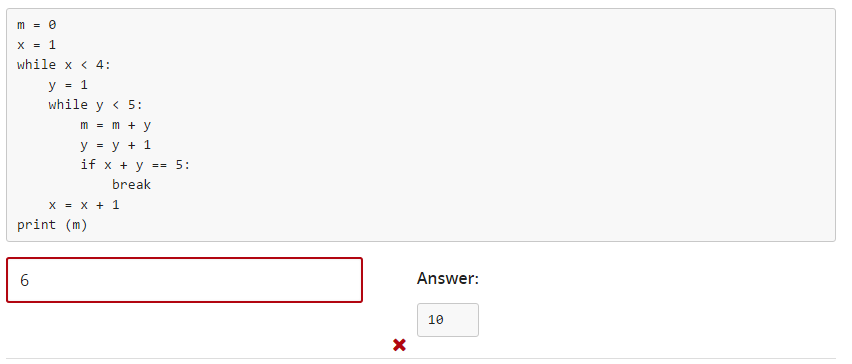
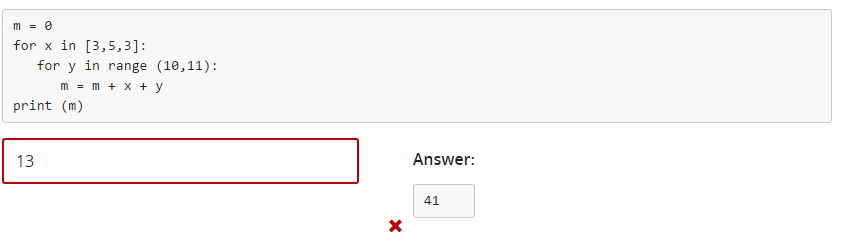
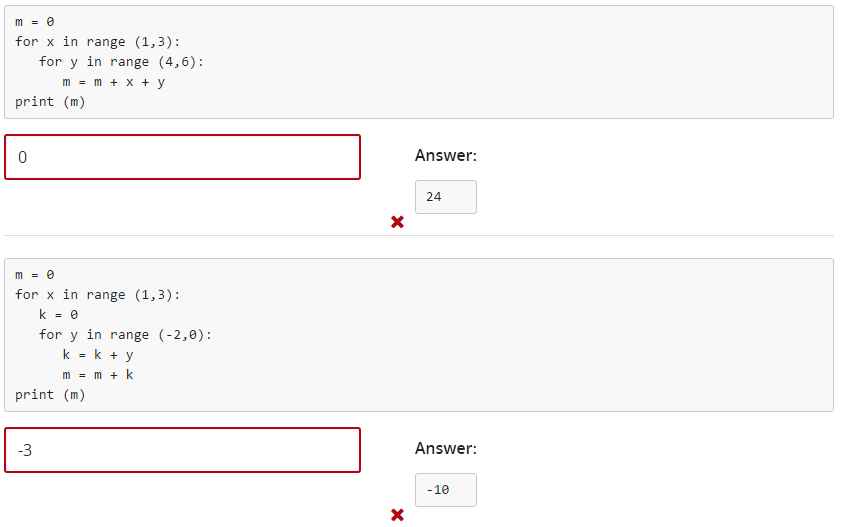
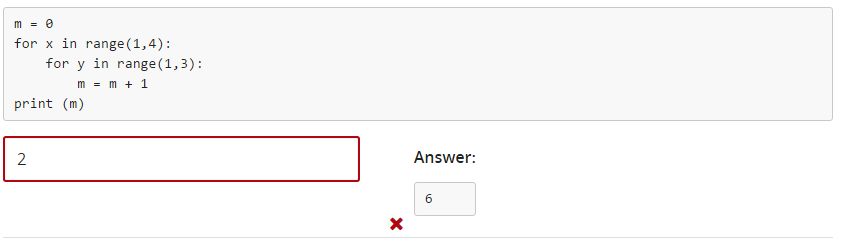


Exercise:





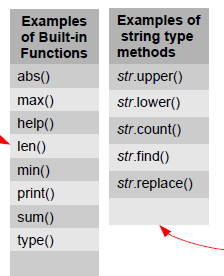


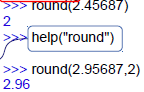


FUNCTIONS

A function is a block of code that performs a specified task. Functions can be thought of as little stand-alone programs that are called and executed within your program and return a specific calculation or carry out a specific task. Many function for common tasks come built into programming languages.





Help on built-in function *round*:

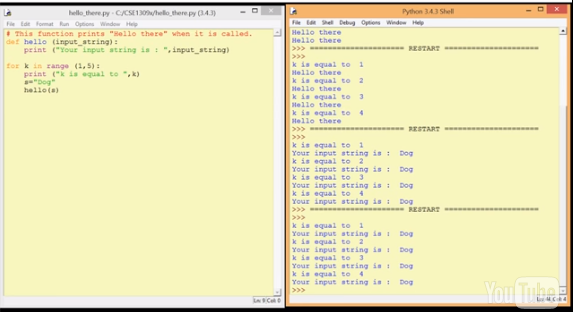
round(...)

round(number[, ndigits]) -> floating point number

Round a number to a given precision in decimal digits (default 0 digits).

This always returns a floating point number. Precision may be negative.

(END)



Most of the time we provide some arguments to a function and it returns something. However, this may not always be the case. While trying to write a function, we need to decide**if the function needs any arguments or not** and **if the function needs to return something or not**. Broadly, we can attempt to classify the combination of function parameters and their return types as follows:

**1. Nothing goes in, nothing comes out.**

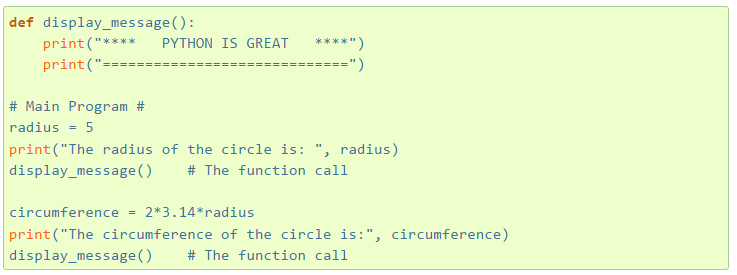
**2. Nothing goes in, something comes out.**

**3. Something goes in, nothing comes out.**

**4. Something goes in, something comes out.**

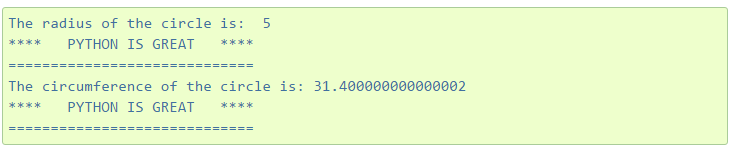
**1. Nothing goes in, nothing comes out.**

Sometimes we need to do some task repeatedly and that task never changes. In this case, the function does not need any arguments (parameters) and it does not need to return anything.



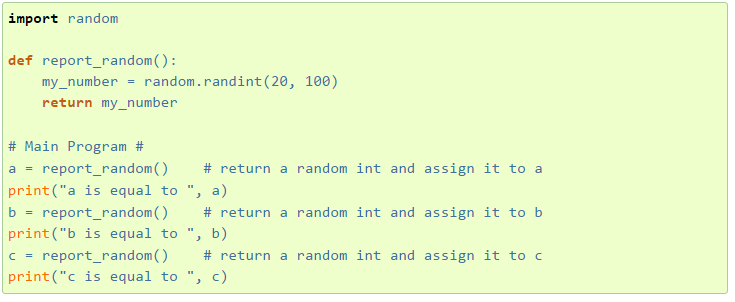
Note that the parentheses in the function call do not have any anything which means the function does not take any arguments. Also nothing is set equal to the function call i.e. the function call is on a line by itself which means it does not need to return anything. So basically, this function just keeps printing the same message every time it is called (Printing something is **not** the same as returning something). Even though, there is no **return** keyword in this function, it returns a **None**by default.

This program produces:



**2. Nothing goes in, something comes out.**

A great example of a function that does not receive anything but does return something is the following.



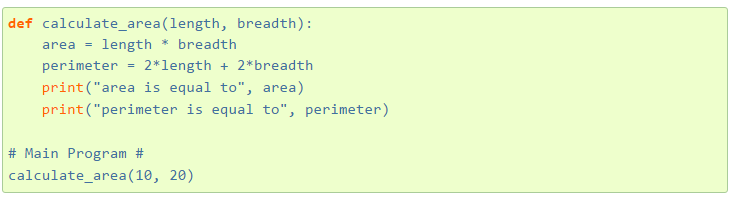
Notice that this function does not receive any arguments but each time we call this function it returns a random integer between 20 and 100 and assigns the returned value to the variable the function call was set equal to.

Sample run of this program produced:



**3. Something goes in, nothing comes out.**

In this scenario, the function needs some arguments to do some task but it does not need to return anything.



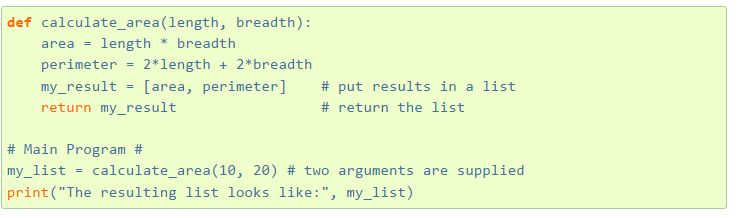
Notice that this function does receive two arguments length and breadth and when we call this function, it calculates the area and perimeter of a rectangle and prints the results but it does not return anything! And again printing something is not the same as returning some value. The return value once again for this function is **None** as there is no **return** keyword.

Sample run of this program produces:



**4. Something goes in, something comes out.**

This is probably the most meaningful scenario where the function takes some arguments and performs some task using those arguments and returns some result as well.



Sample run of this program produces:

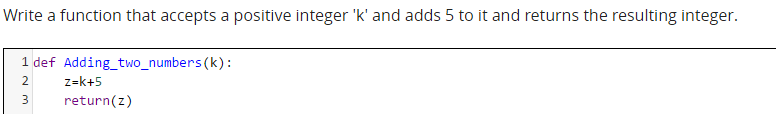


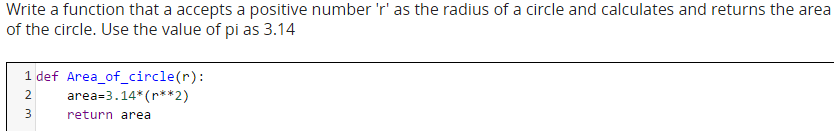
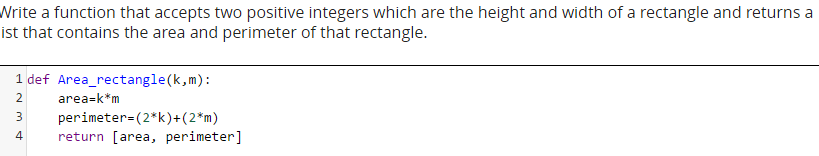
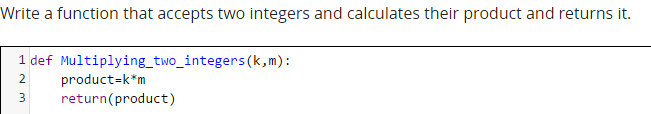
Note that we can return multiple things in python by separating them with a comma. For example instead of returning the results in a list we could have done the following:



In this situation, python would have returned the two values as a 'tuple', which would have looked like:

(200, 60)



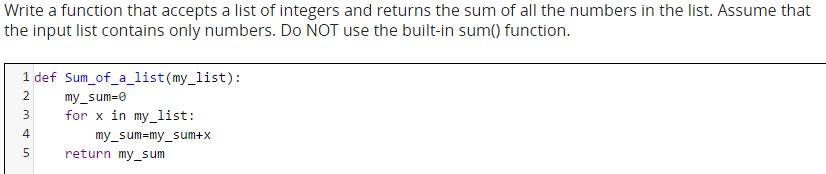


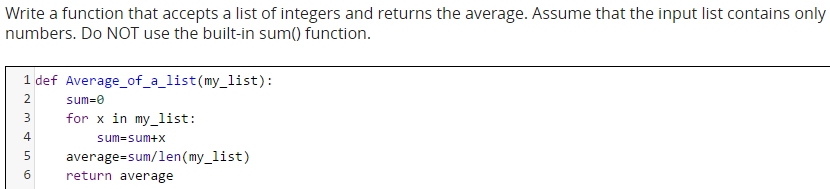
Functions analogy

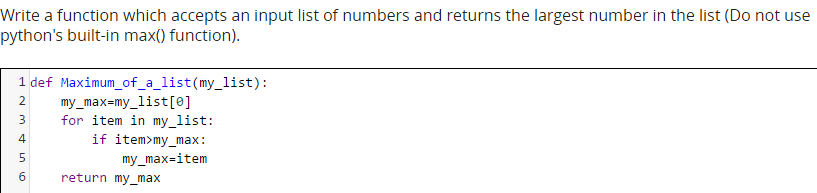
-Functions make a program easier to read and understand

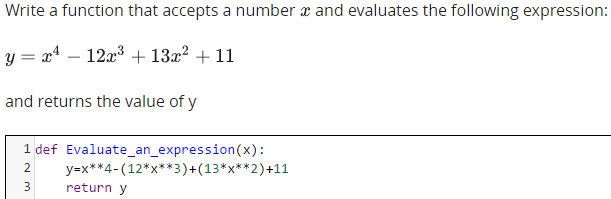
-Functions reduce code duplications

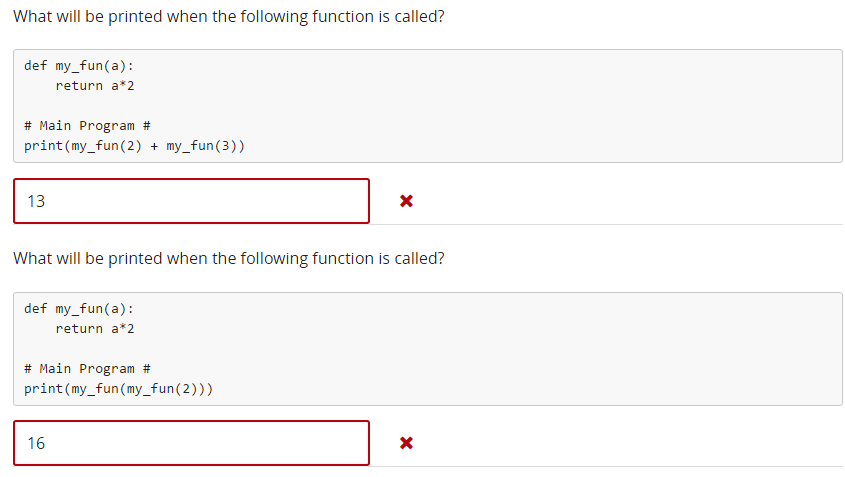
-Functions allow the code to be reused







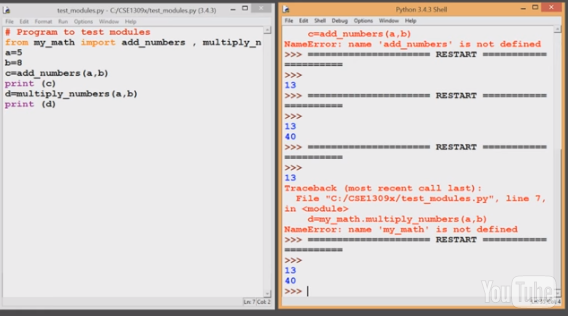




MODULES

Modules are Python functions which are saved in a file with .py extensions.

You can save your function in a file with .py extension and then use the import statement to use that file (module).



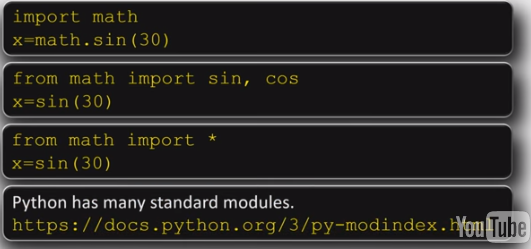
If you use \* after import, that means you are importing ALL the functions within a .py file.

A module can contain executable statements as well as function definitions.

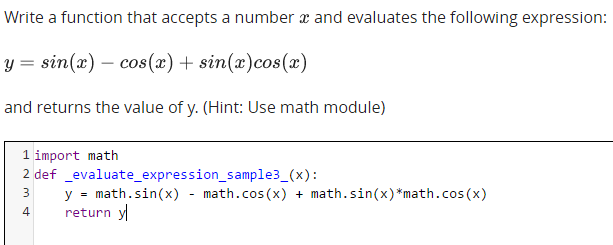
To include a module in your program use:

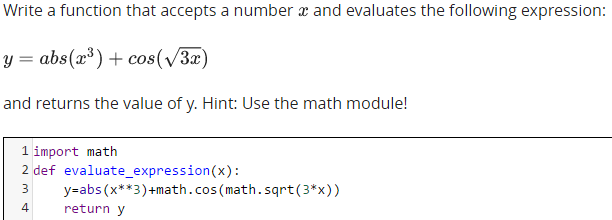


Formats:

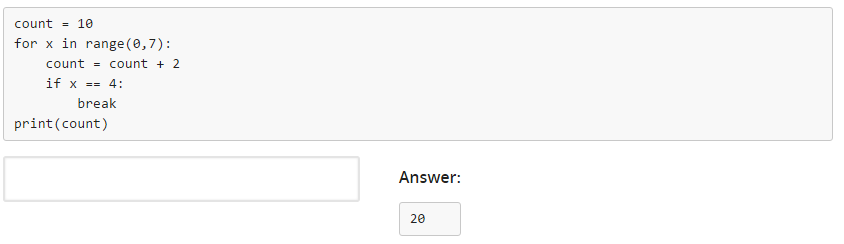


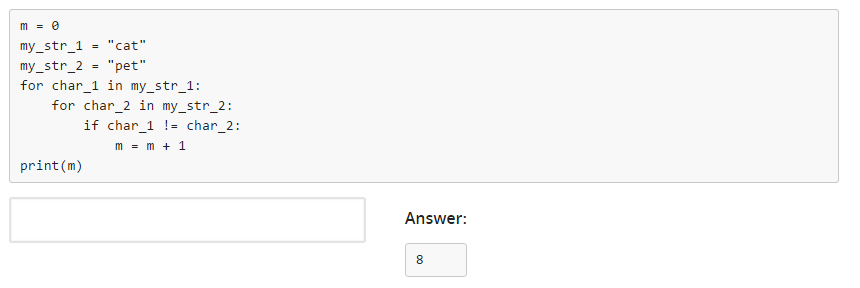
Exercises:

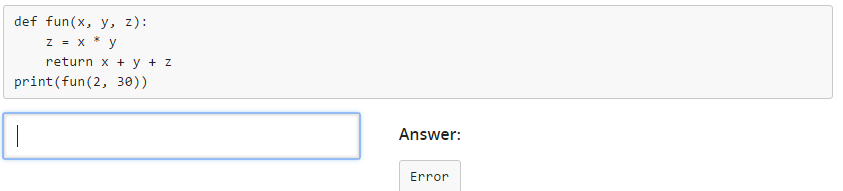




QUIZ 3







FILE Input/Output

Before you can read or write into a file you have to prepare the file and tell Python what is the format of the file and how you intend to use it (read, write, append). This is done by using the *open ()* function.

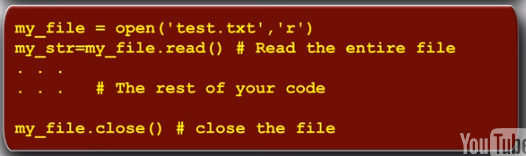


Example:

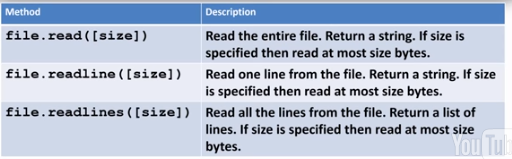


Once the file is open, you can read from the file by using the *read ()* method.

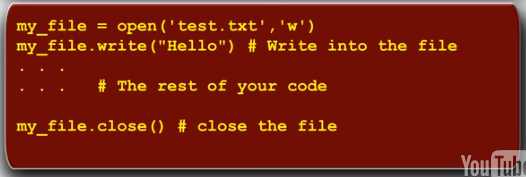
Example:



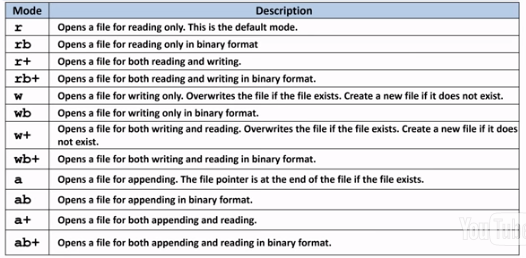
There are three ways you can read from a file:



To write into a file we use the *write ()* method.



File Access Modes

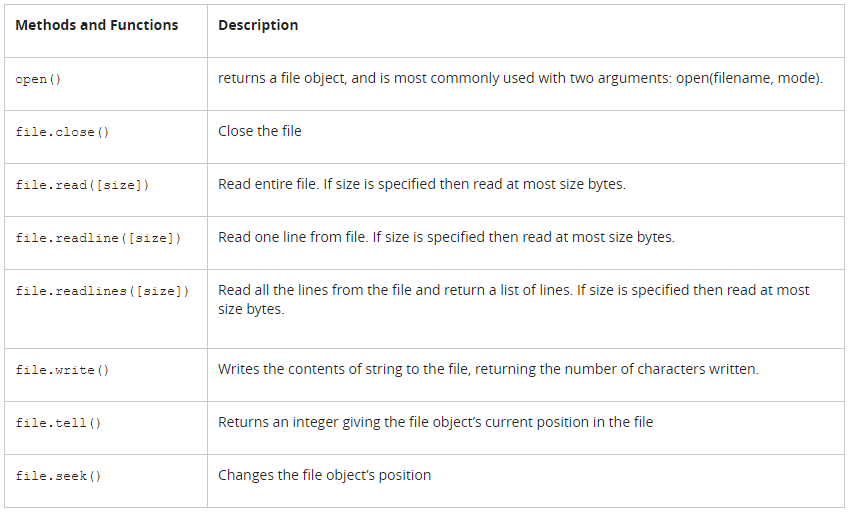


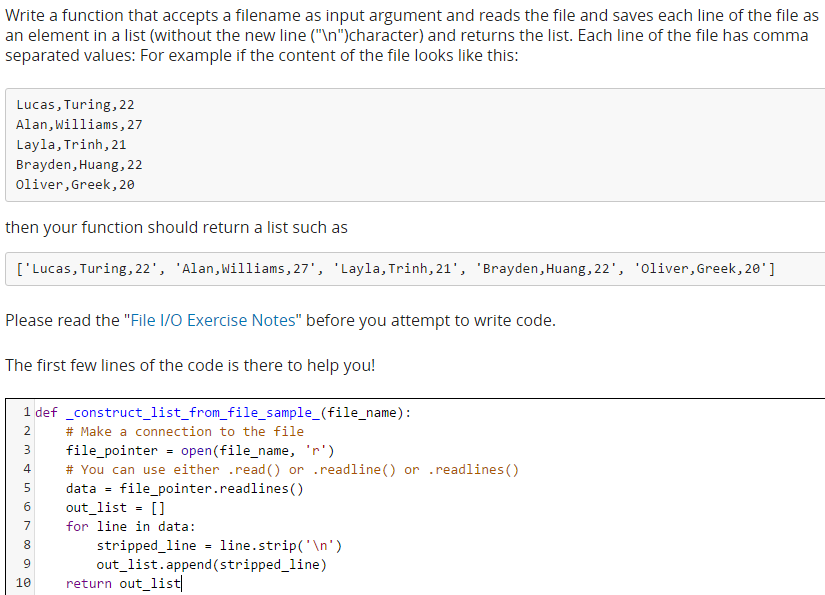
Append mode does not allow the user to read the file.

File Position



Resumen:





Parsing data

1)Removing end-of line chacters with the .strip() method:

a) Run again the script “Unit3\_OI\_example1.py”

b) Edit the file to add the strip() method (line 10 of code): Line=MyFile.readline().strip()

c) Save it as “Unit3\_OI\_example1\_v2.py” and run it.

d) Compare output of both versions

2)Dividing input line into smaller pieces of information (character-delimited files) with the .split() method

a) Open “Unit3\_OI\_example2\_v2.py” in a text editor study the code

b) Run “Unit3\_OI\_example2\_v2.py” and observe the output, compare it with the output

produced by “Unit3\_OI\_example2.py”

c) Edit “Unit3\_OI\_example2\_v2.py” so it reads the file aa\_frequencies\_v2.csv and prints only

the second column. Save it as “Unit3\_OI\_example2\_v3.py” and run it.

Be aware of two common types of data files:

1. Table-like

2. “keyword” based

HANDLING THE FILESYSTEM

