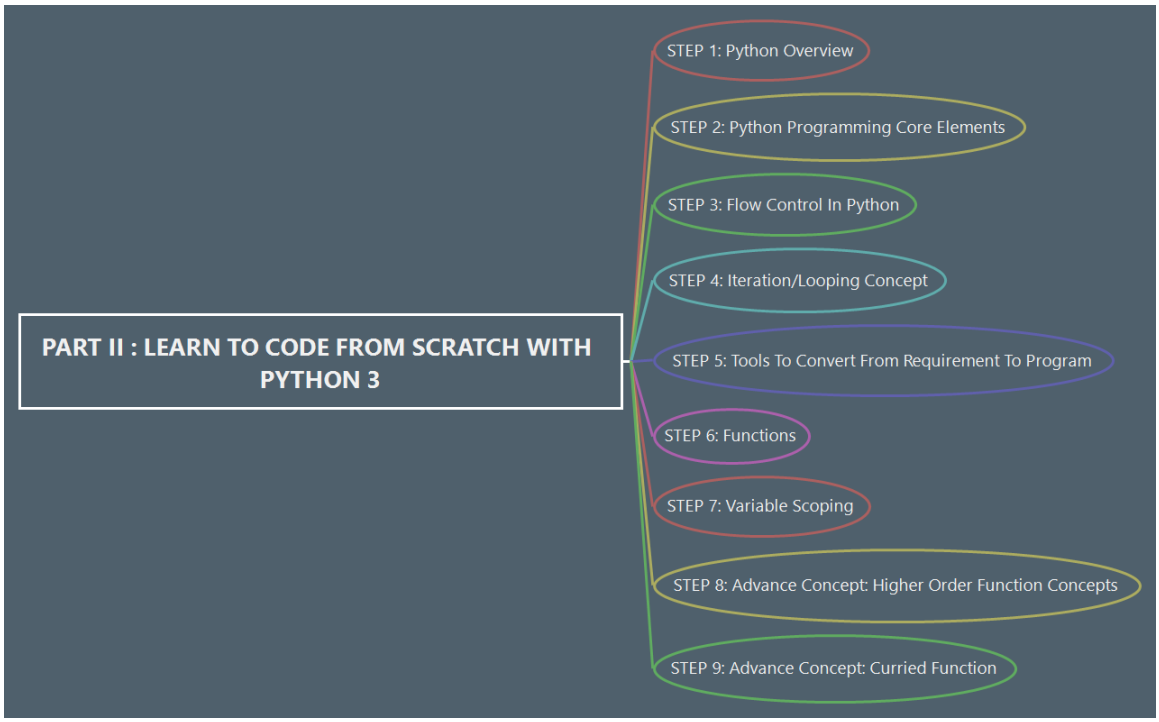
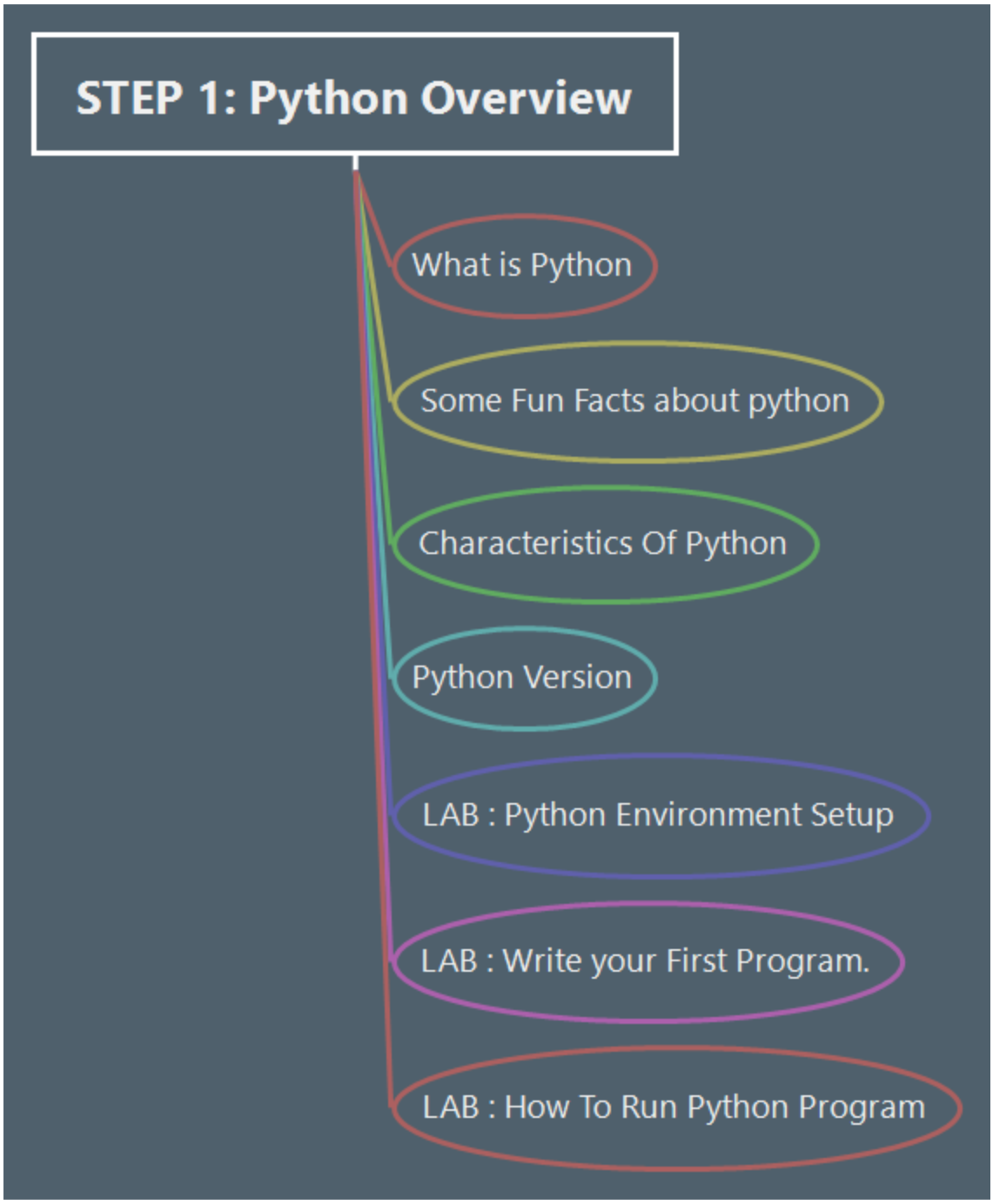
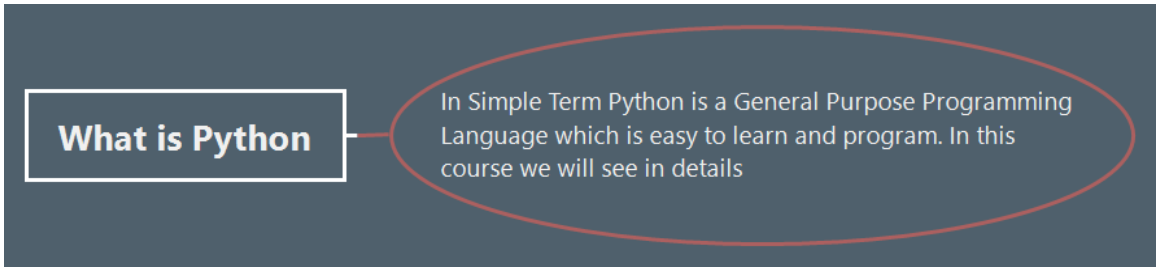
PART II : LEARN TO CODE FROM SCRATCH WITH PYTHON 3



1. STEP 1: Python Overview

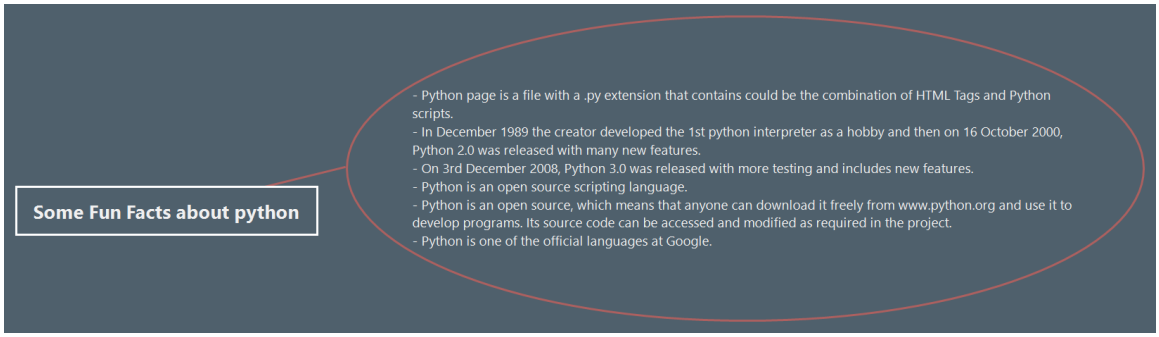


1.1. What is Python



1.1.1. In Simple Term Python is a General Purpose Programming Language which is easy to learn and program. In this course we will see in details

1.2. Some Fun Facts about python



1.2.1. - Python page is a file with a .py extension that contains could be the combination of HTML Tags and Python scripts.

- In December 1989 the creator developed the 1st python interpreter as a hobby and then on 16 October 2000, Python 2.0 was released with many new features.

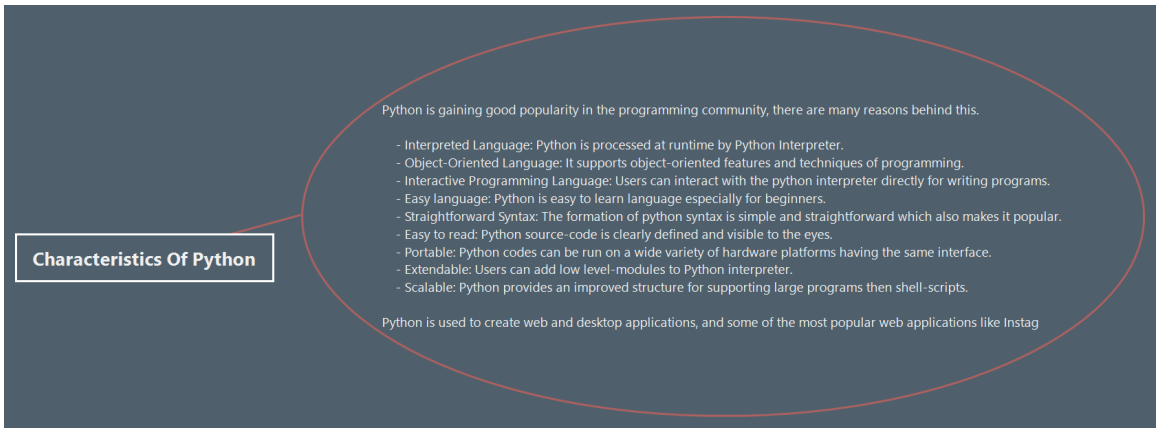
- On 3rd December 2008, Python 3.0 was released with more testing and includes new features.

- Python is an open source scripting language.

- Python is an open source, which means that anyone can download it freely from www.python.org and use it to develop programs. Its source code can be accessed and modified as required in the project.

- Python is one of the official languages at Google.

1.3. Characteristics Of Python



1.3.1. Python is gaining good popularity in the programming community, there are many reasons behind this.

- Interpreted Language: Python is processed at runtime by Python Interpreter.

- Object-Oriented Language: It supports object-oriented features and techniques of programming.

- Interactive Programming Language: Users can interact with the python interpreter directly for writing programs.

- Easy language: Python is easy to learn language especially for beginners.

- Straightforward Syntax: The formation of python syntax is simple and straightforward which also makes it popular.

- Easy to read: Python source-code is clearly defined and visible to the eyes.

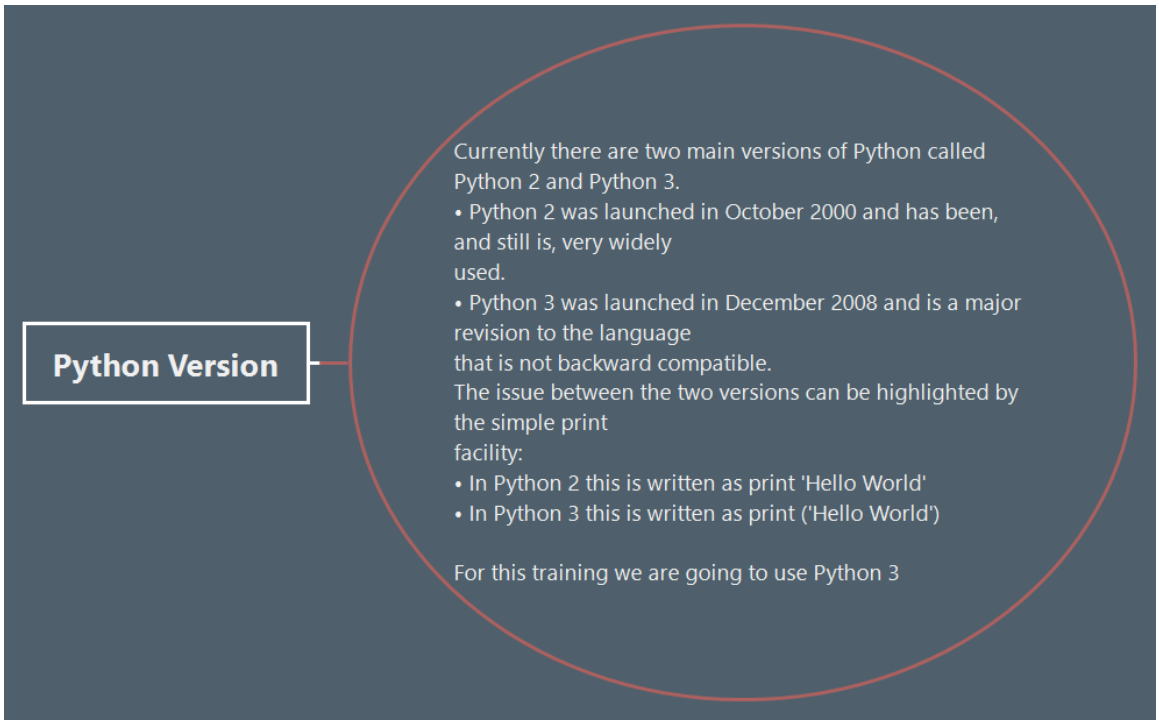
- Portable: Python codes can be run on a wide variety of hardware platforms having the same interface.

- Extendable: Users can add low level-modules to Python interpreter.

- Scalable: Python provides an improved structure for supporting large programs then shell-scripts.

Python is used to create web and desktop applications, and some of the most popular web applications like Instag

1.4. Python Version



1.4.1. Currently there are two main versions of Python called Python 2 and Python 3.

• Python 2 was launched in October 2000 and has been, and still is, very widely used.

• Python 3 was launched in December 2008 and is a major revision to the language

that is not backward compatible.

The issue between the two versions can be highlighted by the simple print facility:

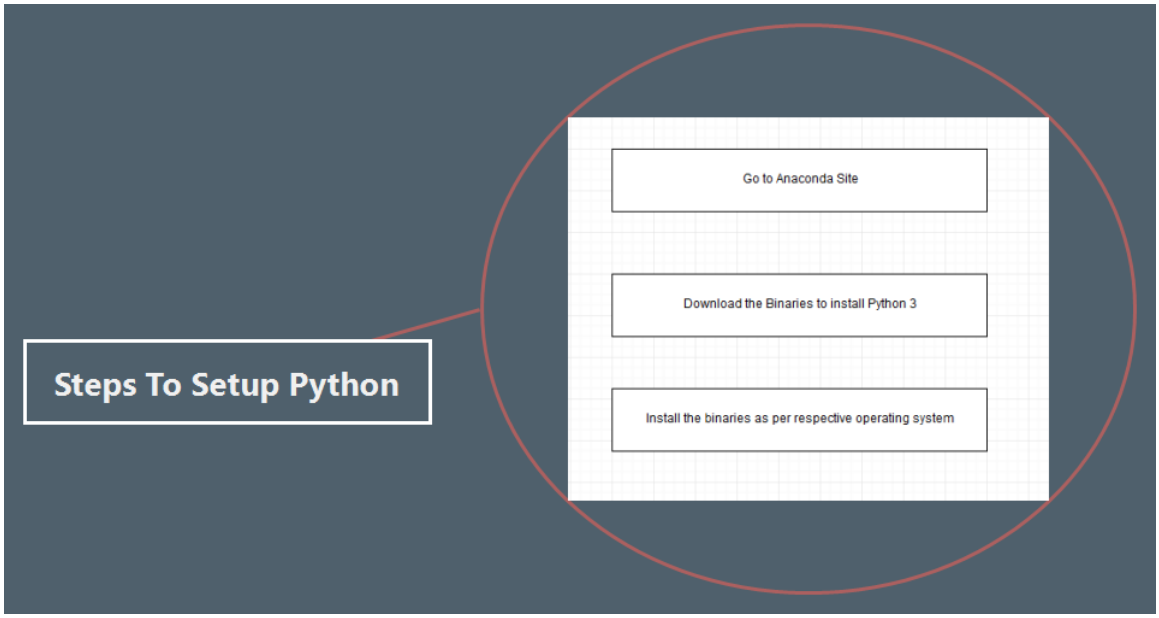
• In Python 2 this is written as print 'Hello World'

• In Python 3 this is written as print ('Hello World')

For this training we are going to use Python 3

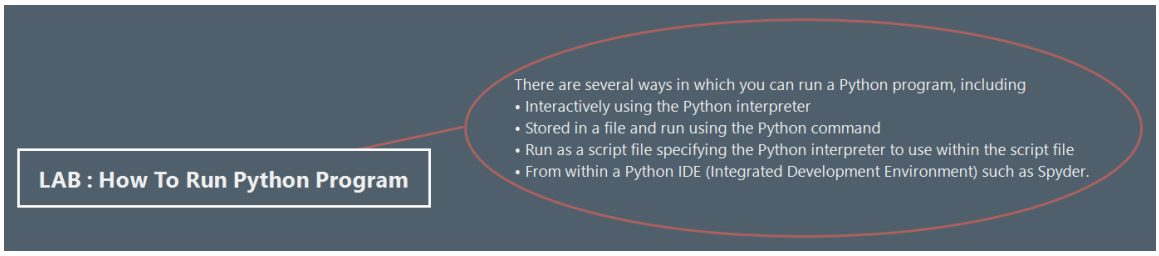
1.5. LAB : Python Environment Setup

1.5.1. Steps To Setup Python



1.6. LAB : Write your First Program.

1.7. LAB : How To Run Python Program

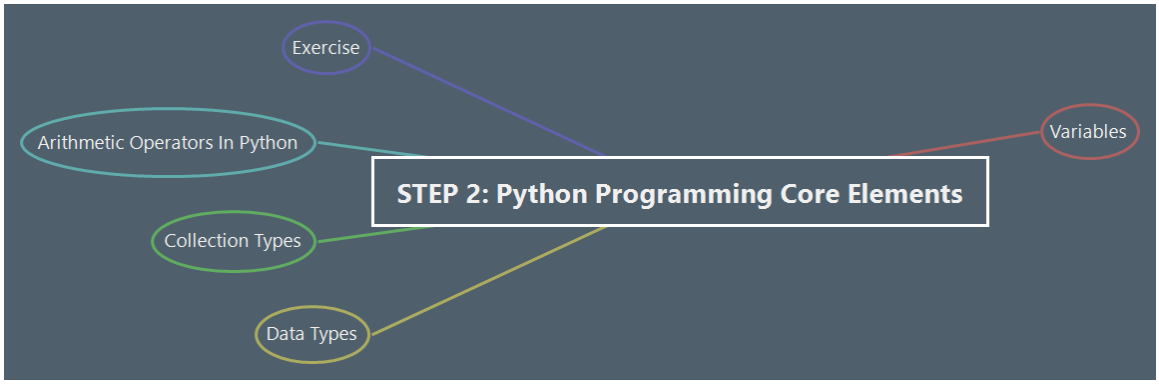


1.7.1. There are several ways in which you can run a Python program, including

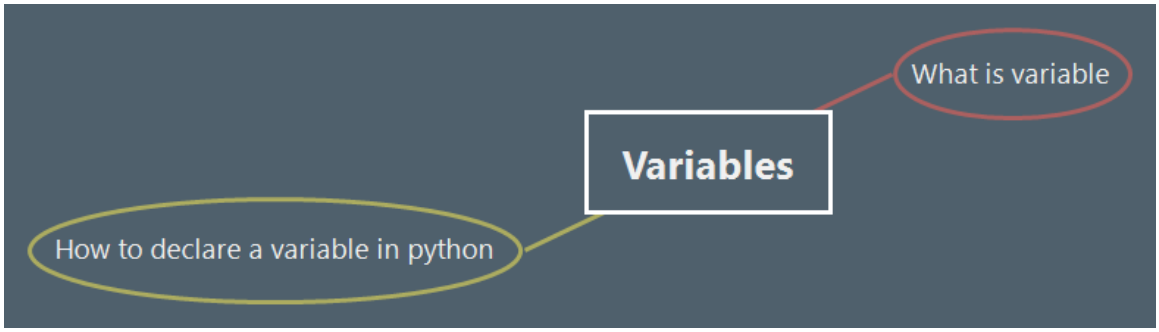
- **Interactively using the Python interpreter**
- **Stored in a file and run using the Python command**

- **Run as a script file specifying the Python interpreter to use within the script file**
- **From within a Python IDE (Integrated Development Environment) such as Spyder.**

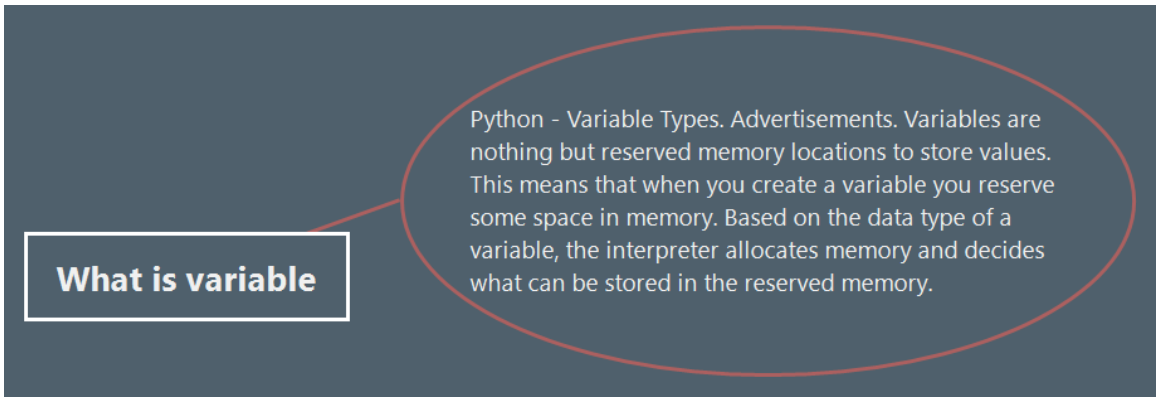
2. STEP 2: Python Programming Core Elements



2.1. Variables

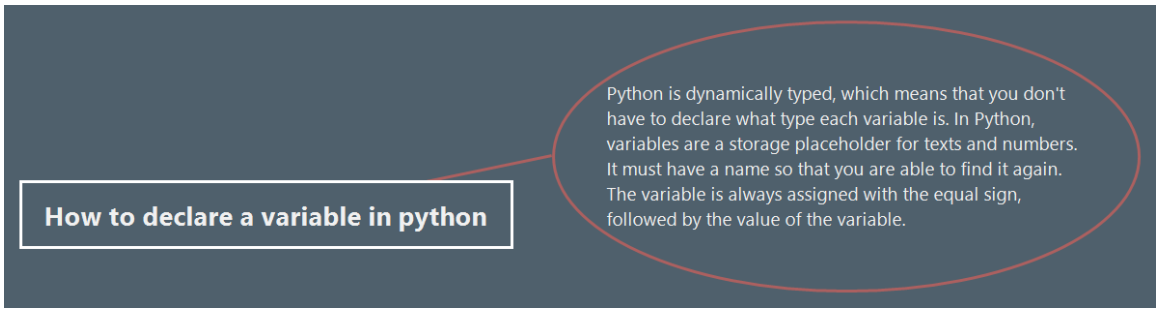


2.1.1. What is variable



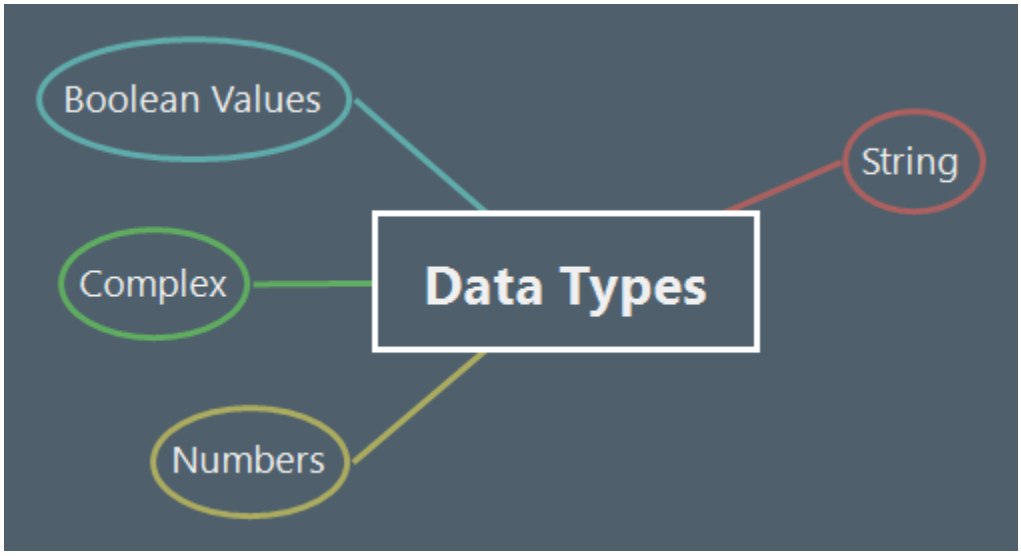
Python - Variable Types. Advertisements. Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.

2.1.2. How to declare a variable in python

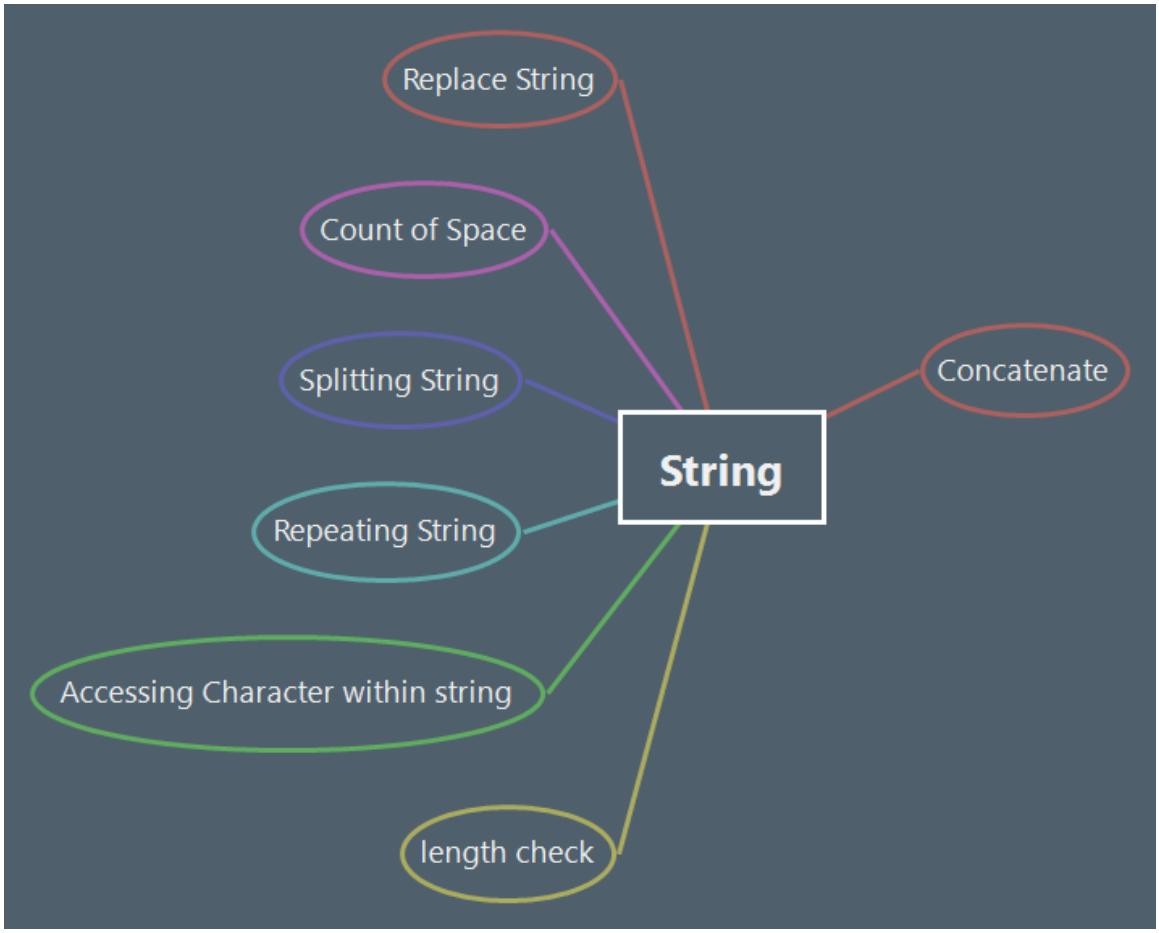


Python is dynamically typed, which means that you don't have to declare what type each variable is. In Python, variables are a storage placeholder for texts and numbers. It must have a name so that you are able to find it again. The variable is always assigned with the equal sign, followed by the value of the variable.

2.2. Data Types



2.2.1. String



Concatenate

length check

Accessing Character within string

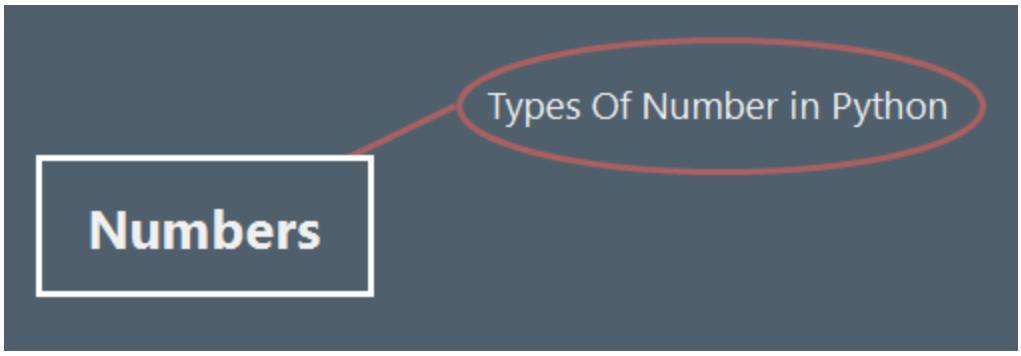
Repeating String

Splitting String

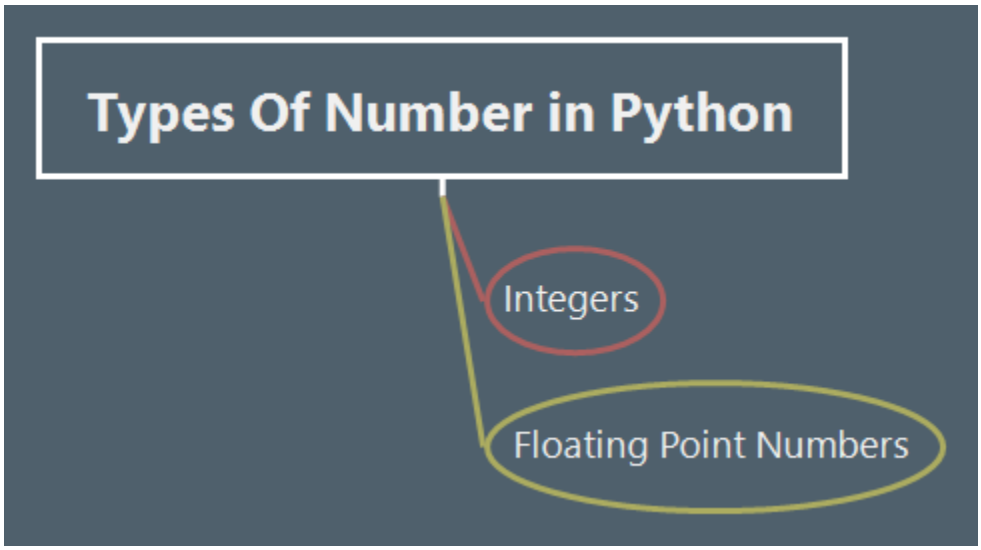
Count of Space

Replace String

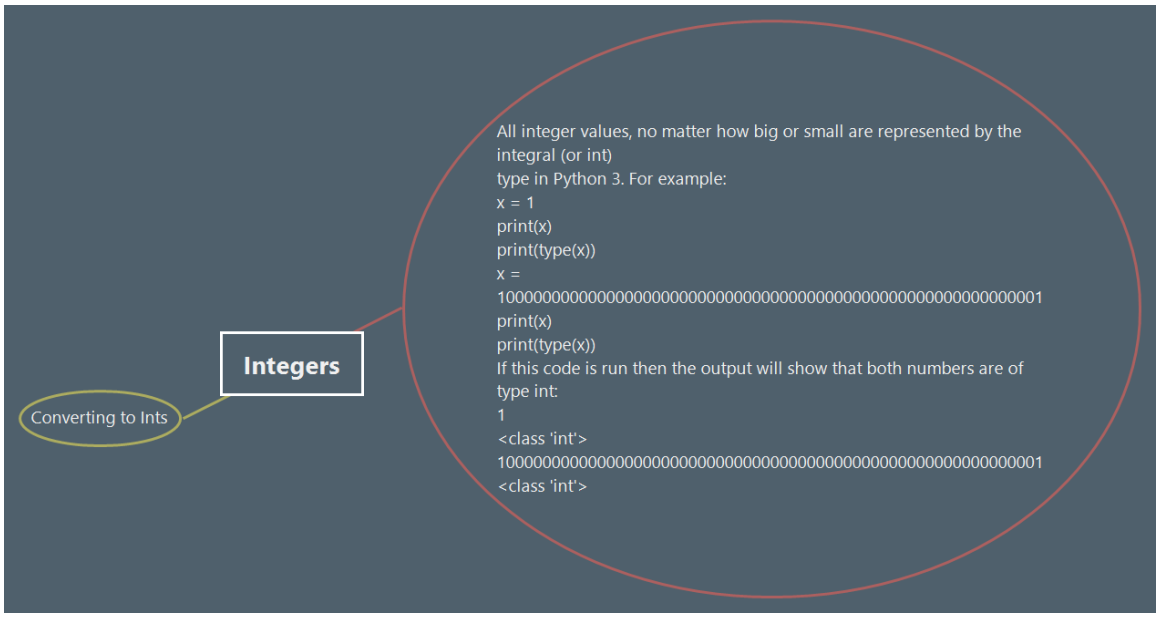
2.2.2. Numbers



Types Of Number in Python



Integers



All integer values, no matter how big or small are represented by the integral (or int)

type in Python 3. For example:

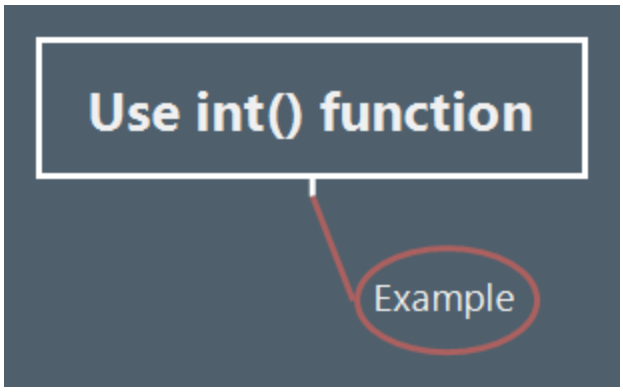
```
x = 1
print(x)
print(type(x))
x =
10000000000000000000000000000000000000000000000000000000001
print(x)
print(type(x))
```

If this code is run then the output will show that both numbers are of type int:

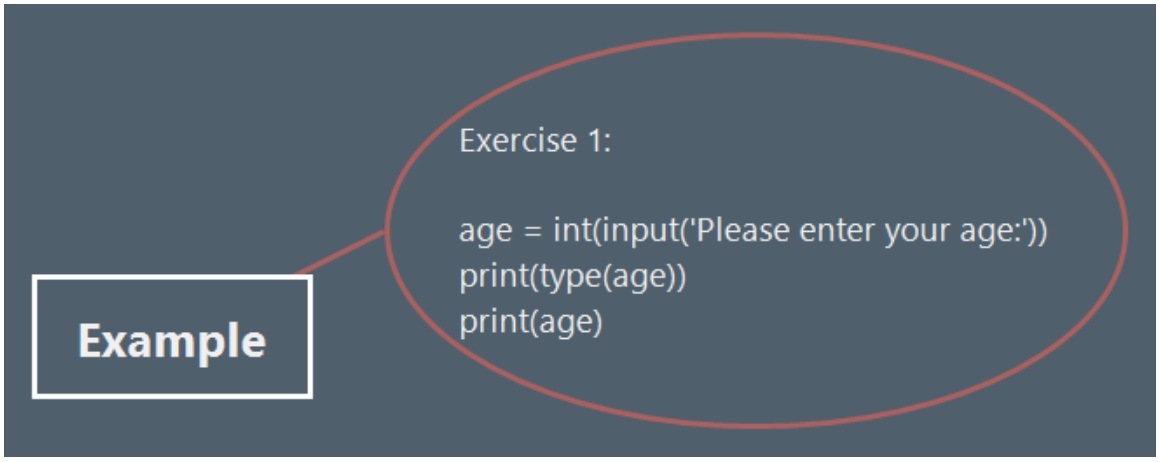
```
1
<class 'int'>
10000000000000000000000000000000000000000000000000000000001
<class 'int'>
```

Converting to Ints

Use int() function



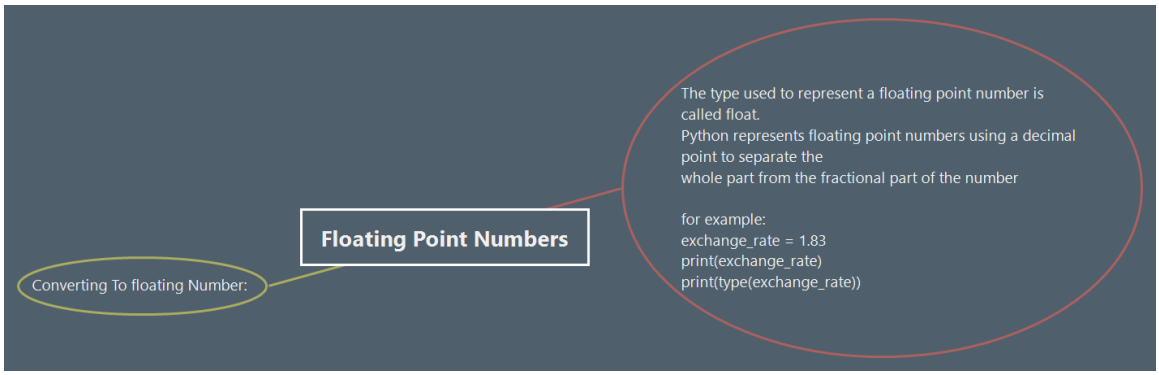
Example



Exercise 1:

```
age = int(input('Please enter your age:'))
print(type(age))
print(age)
```

Floating Point Numbers



The type used to represent a floating point number is called float.

Python represents floating point numbers using a decimal point to separate the

whole part from the fractional part of the number

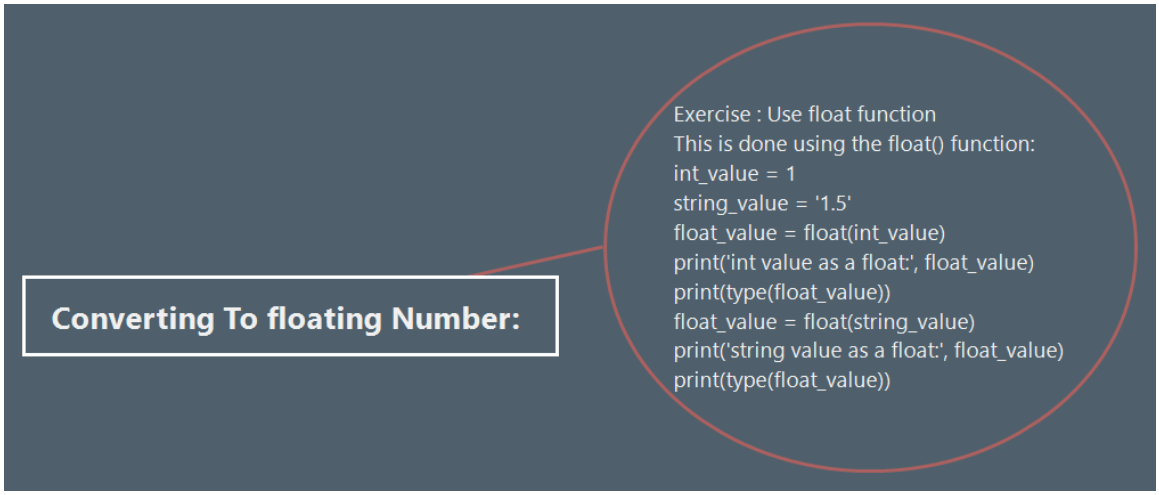
for example:

exchange_rate = 1.83

print(exchange_rate)

print(type(exchange_rate))

Converting To floating Number:



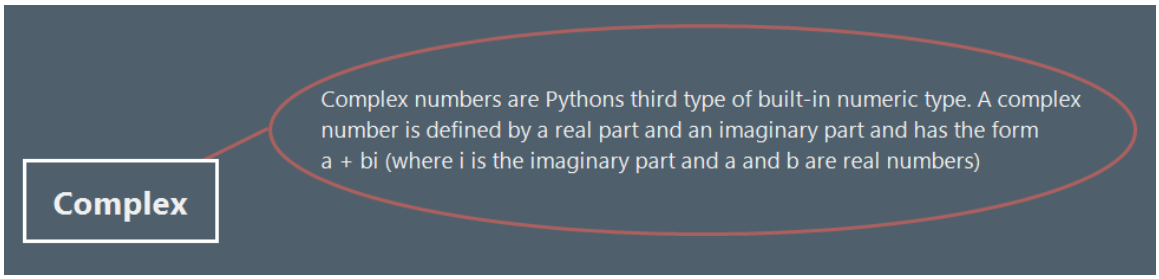
Exercise : Use float function

This is done using the float() function:

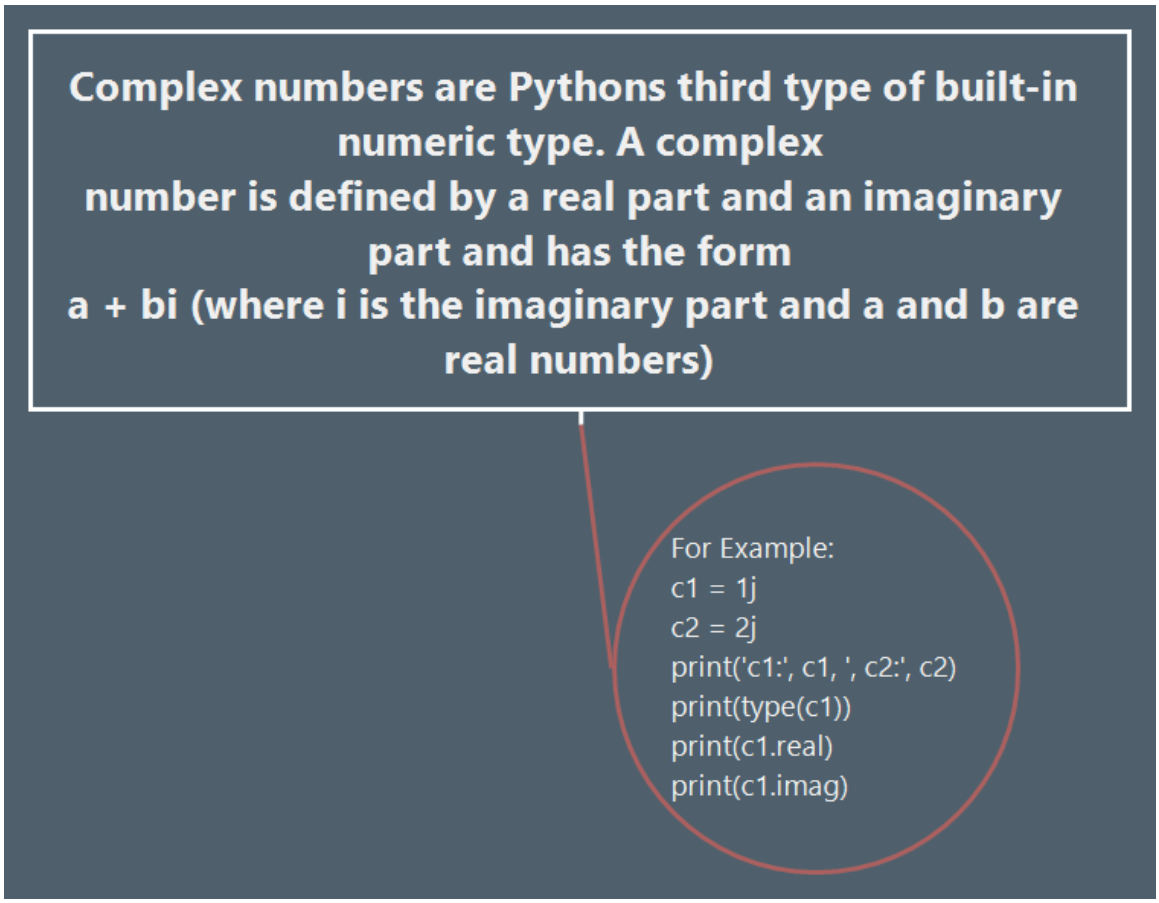
int_value = 1

```
string_value = '1.5'
float_value = float(int_value)
print('int value as a float:', float_value)
print(type(float_value))
float_value = float(string_value)
print('string value as a float:', float_value)
print(type(float_value))
```

2.2.3. Complex



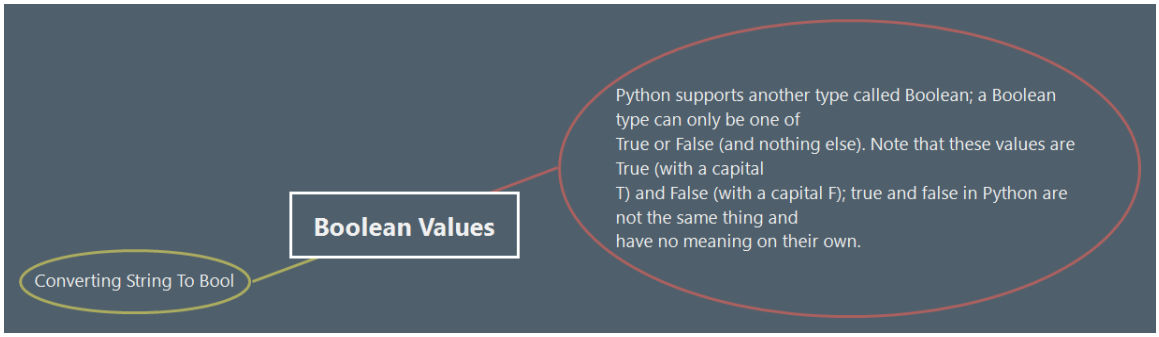
Complex numbers are Pythons third type of built-in numeric type. A complex number is defined by a real part and an imaginary part and has the form a + bi (where i is the imaginary part and a and b are real numbers)



For Example:

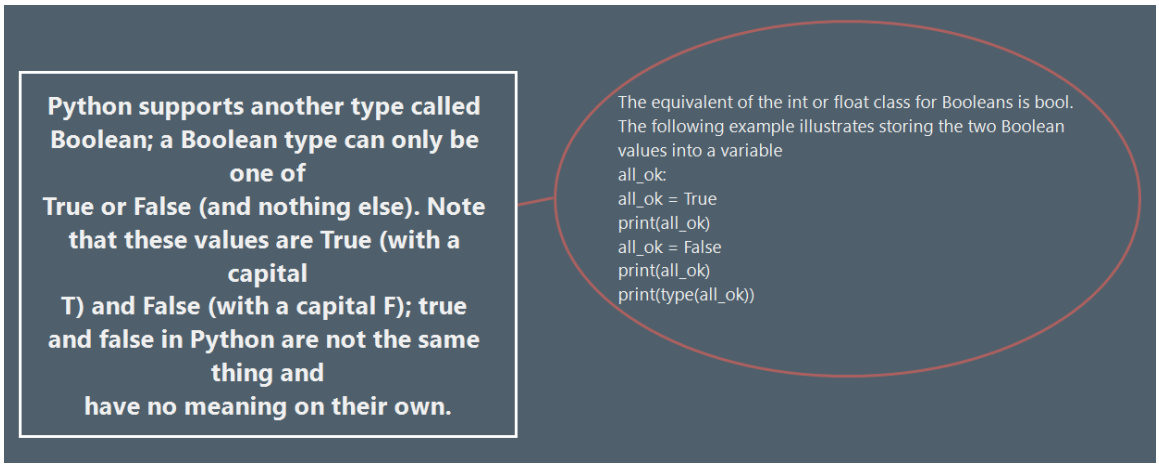
```
c1 = 1j
c2 = 2j
print('c1:', c1, ', c2:', c2)
print(type(c1))
print(c1.real)
print(c1.imag)
```

2.2.4. Boolean Values



Python supports another type called Boolean; a Boolean type can only be one of True or False (and nothing else). Note that these values are True (with a capital T) and False (with a capital F); true and false in Python are not the same thing and

have no meaning on their own.



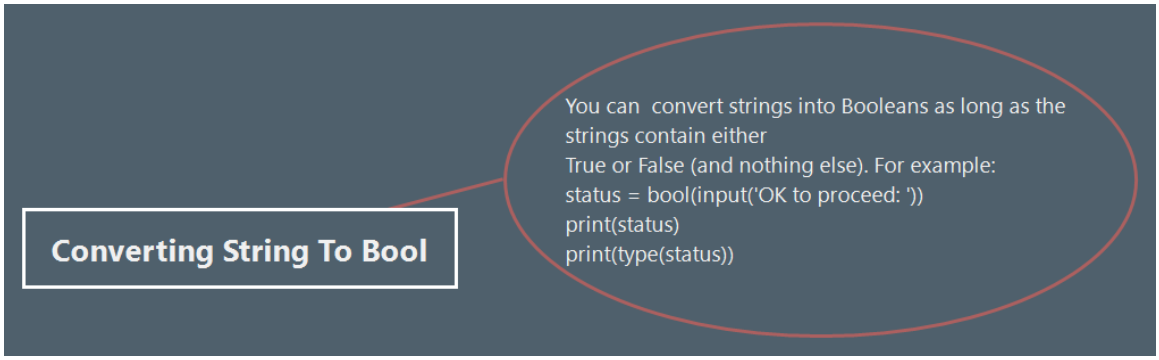
The equivalent of the int or float class for Booleans is bool.

The following example illustrates storing the two Boolean values into a variable

all_ok:

```
all_ok = True
print(all_ok)
all_ok = False
print(all_ok)
print(type(all_ok))
```

Converting String To Bool



You can convert strings into Booleans as long as the strings contain either True or False (and nothing else). For example:

```
status = bool(input('OK to proceed: '))
print(status)
print(type(status))
```

2.3. Collection Types

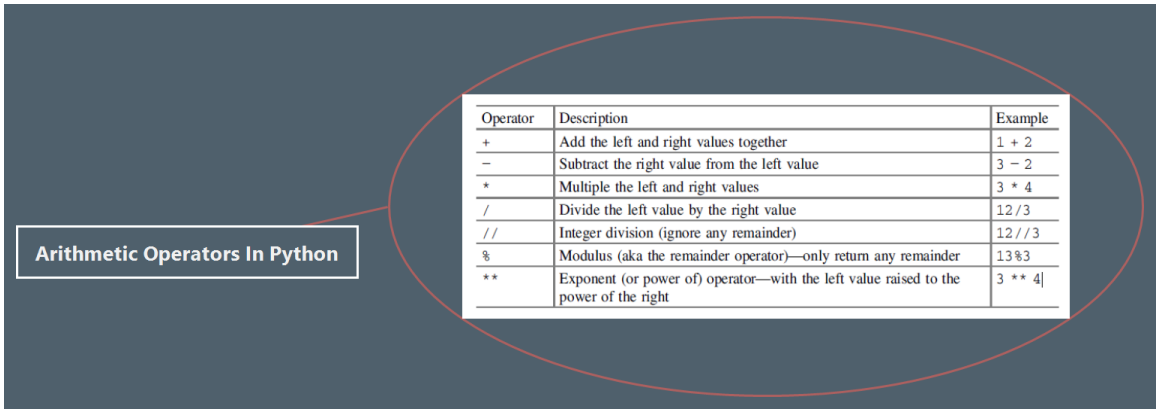
2.3.1. Dictionary

2.3.2. List

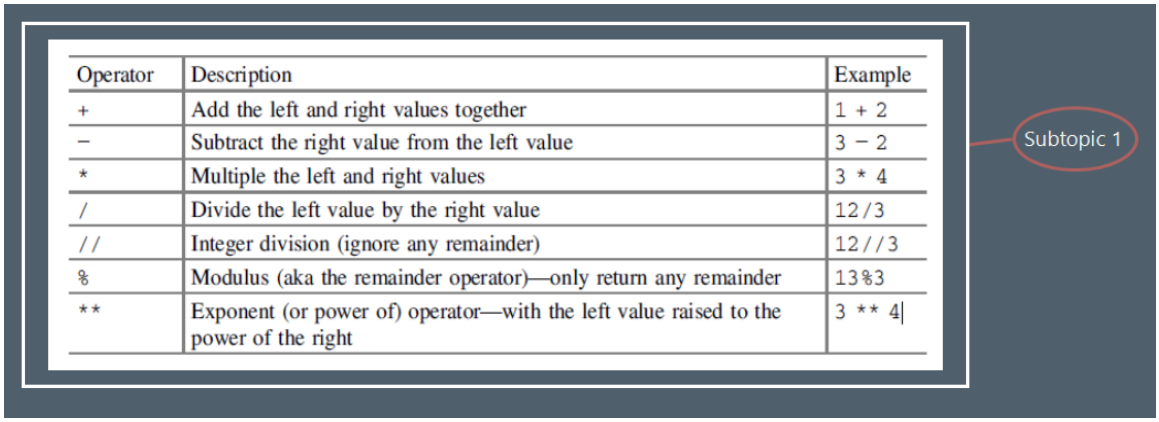
2.3.3. Set

2.3.4. Tuple

2.4. Arithmetic Operators In Python

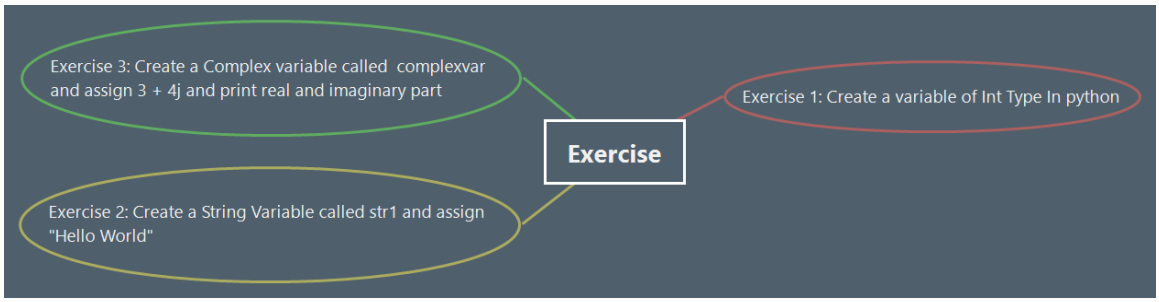


2.4.1.



Subtopic 1

2.5. Exercise

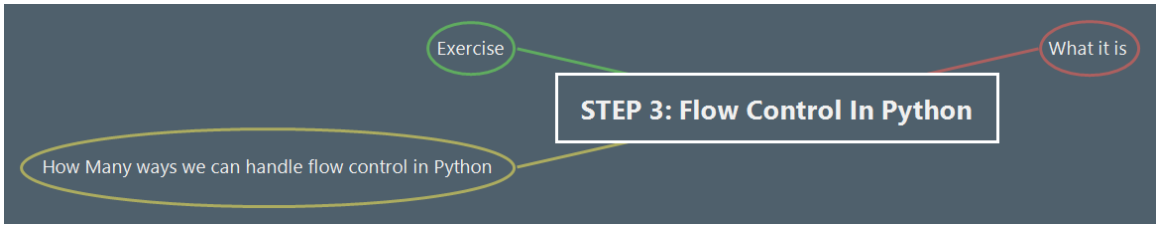


2.5.1. Exercise 1: Create a variable of Int Type In python

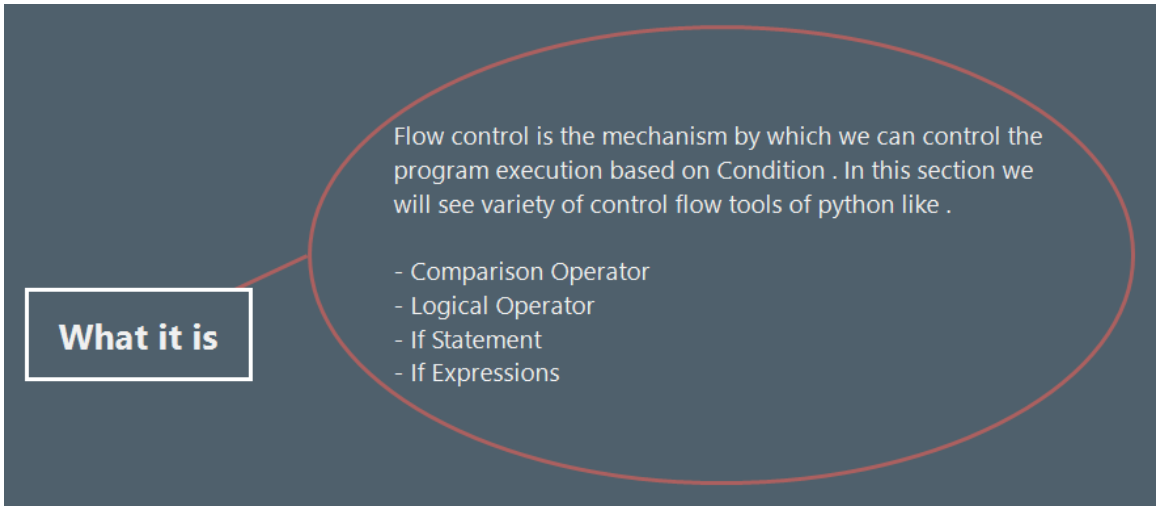
2.5.2. Exercise 2: Create a String Variable called str1 and assign "Hello World"

2.5.3. Exercise 3: Create a Complex variable called complexvar and assign 3 + 4j and print real and imaginary part

3. STEP 3: Flow Control In Python



3.1. What it is



3.1.1. Flow control is the mechanism by which we can control the program execution based on Condition . In this section we will see variety of control flow tools of python like .

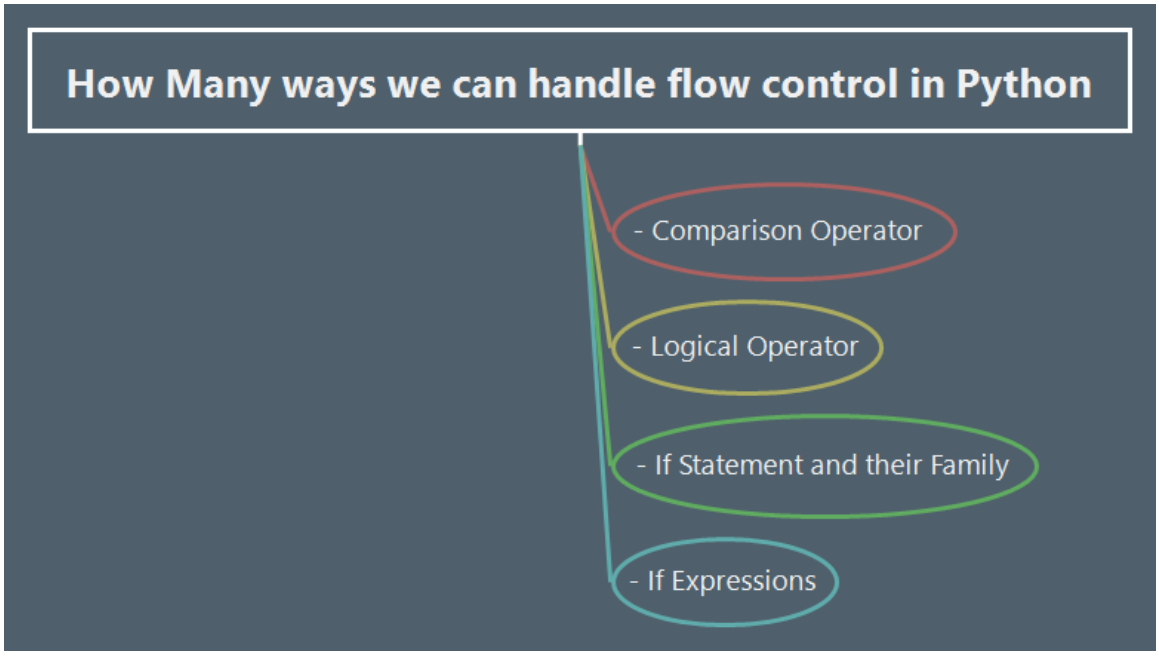
- Comparison Operator

- Logical Operator

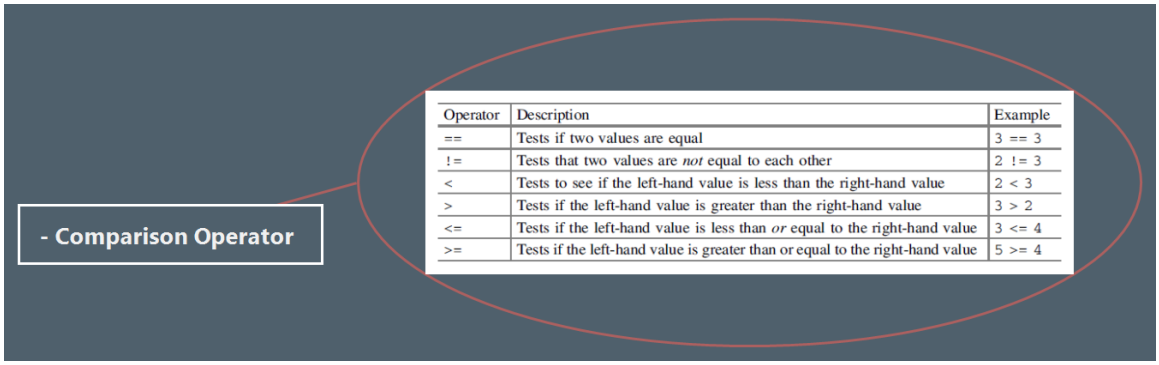
- If Statement

- If Expressions

3.2. How Many ways we can handle flow control in Python

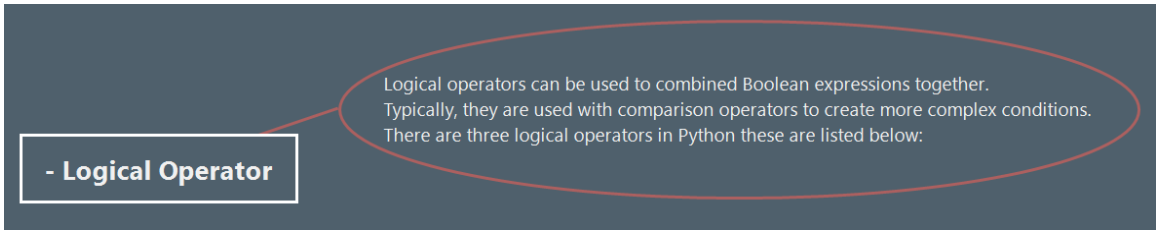


3.2.1. - Comparison Operator



Operator	Description	Example
==	Tests if two values are equal	3 == 3
!=	Tests that two values are *not* equal to each other	2 != 3
<	Tests to see if the left-hand value is less than the right-hand value	2 < 3
>	Tests if the left-hand value is greater than the right-hand value	3 > 2
<=	Tests if the left-hand value is less than *or* equal to the right-hand value	3 <= 4
>=	Tests if the left-hand value is greater than or equal to the right-hand value	5 >= 4

3.2.2. - Logical Operator

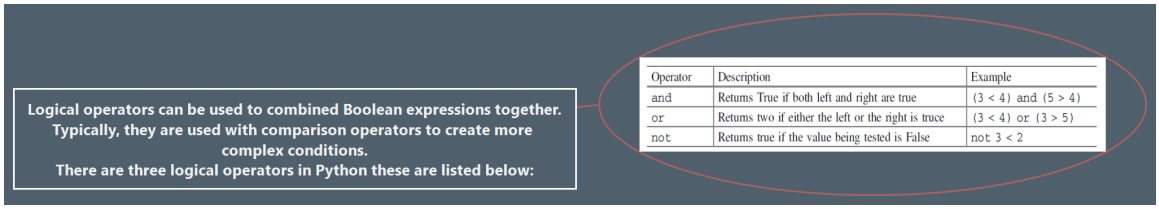


Logical operators can be used to combined Boolean expressions together.

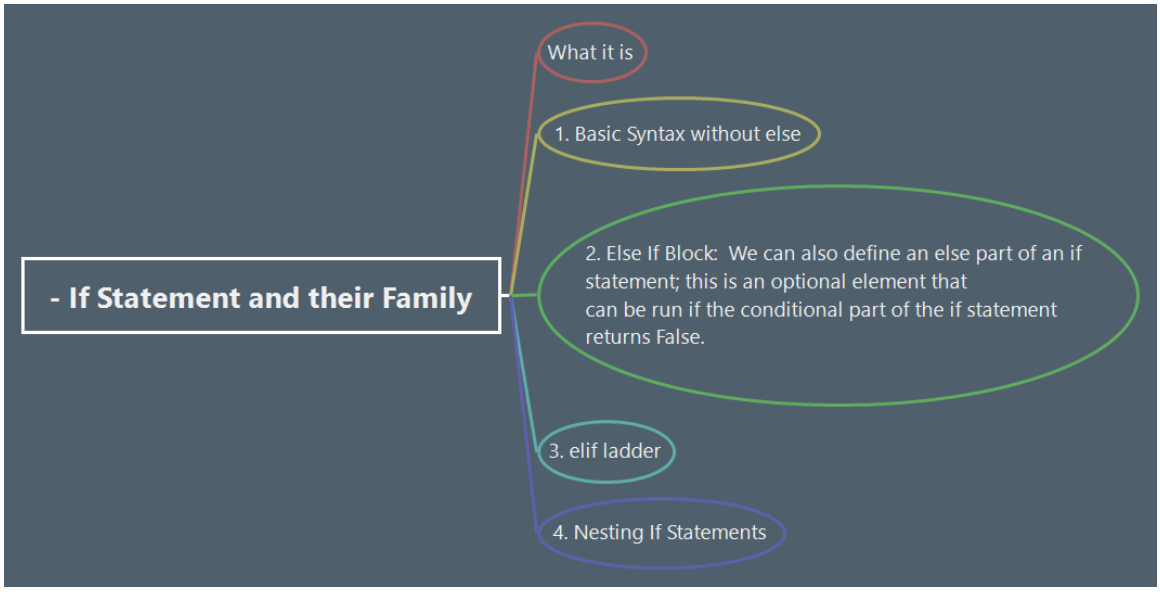
Typically, they are used with comparison operators to create more complex

conditions.

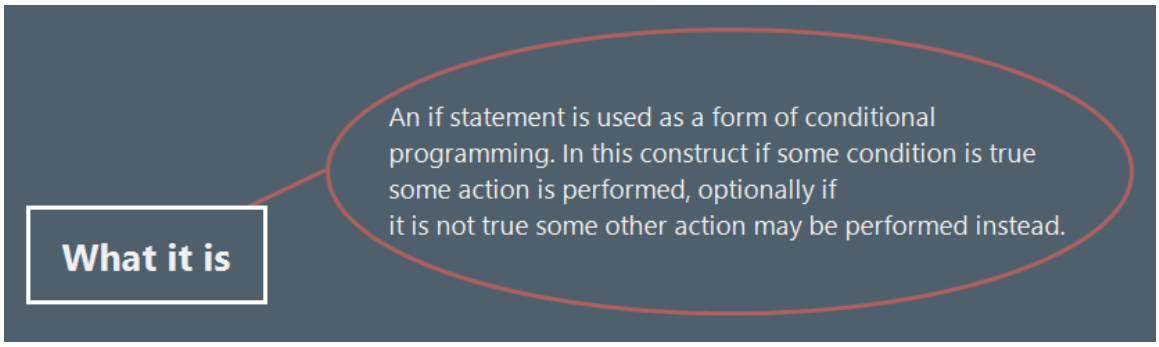
There are three logical operators in Python these are listed below:



3.2.3. - If Statement and their Family

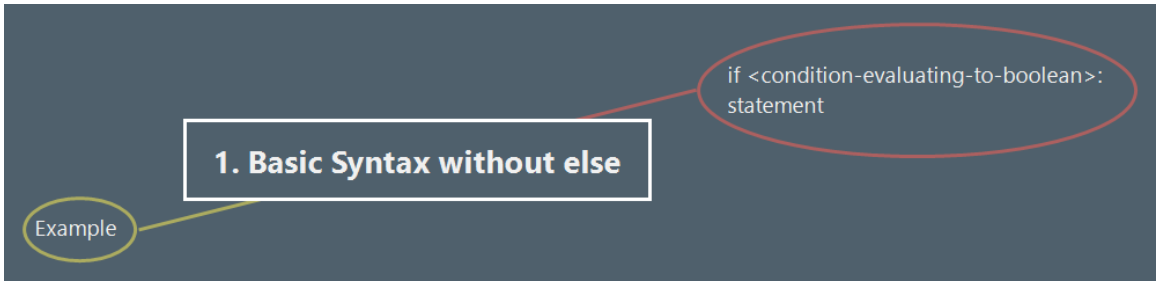


What it is



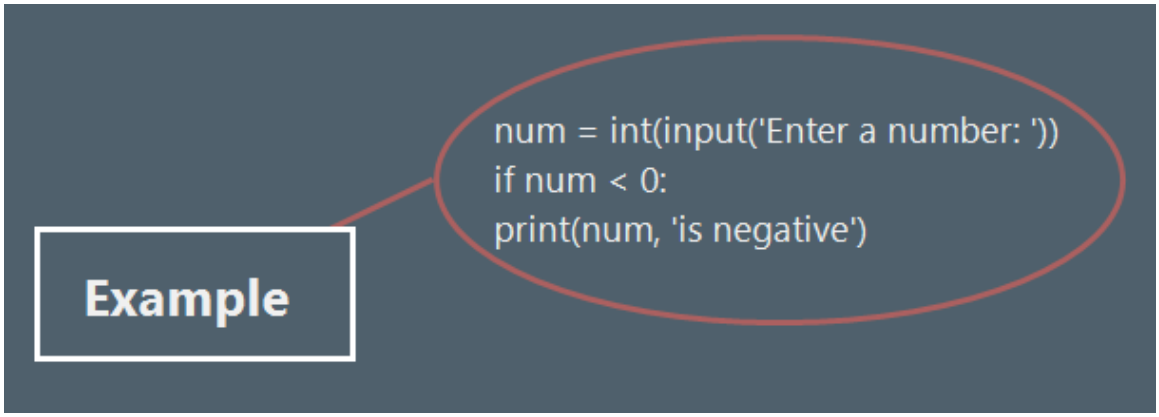
An if statement is used as a form of conditional programming. In this construct if some condition is true some action is performed, optionally if it is not true some other action may be performed instead.

1. Basic Syntax without else



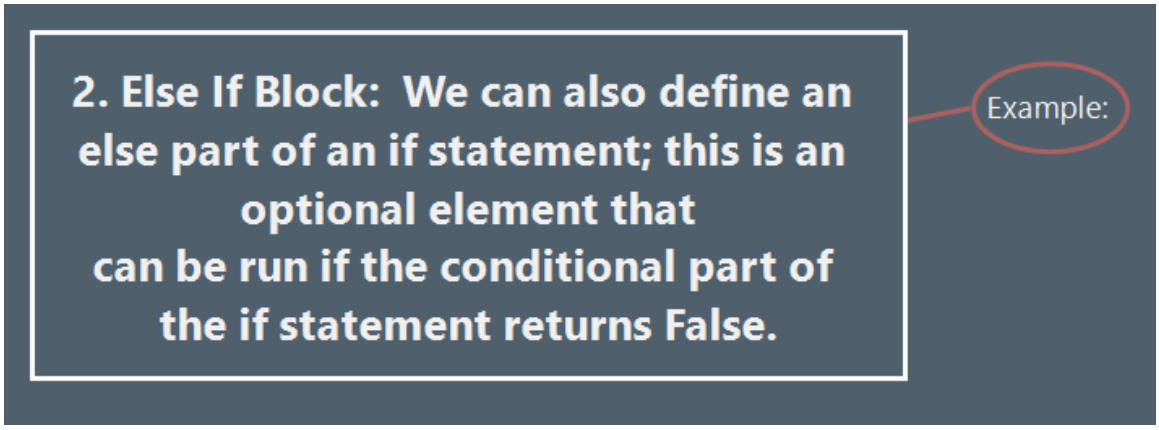
if <condition-evaluating-to-boolean>:
statement

Example

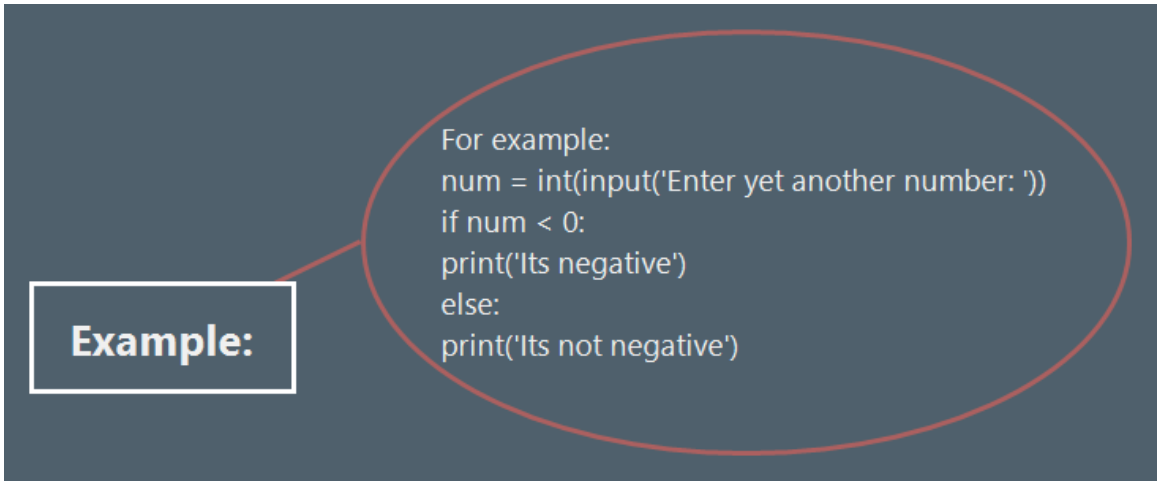



```
num = int(input('Enter a number: '))
if num < 0:
print(num, 'is negative')
```

2. Else If Block: We can also define an else part of an if statement; this is an optional element that can be run if the conditional part of the if statement returns False.



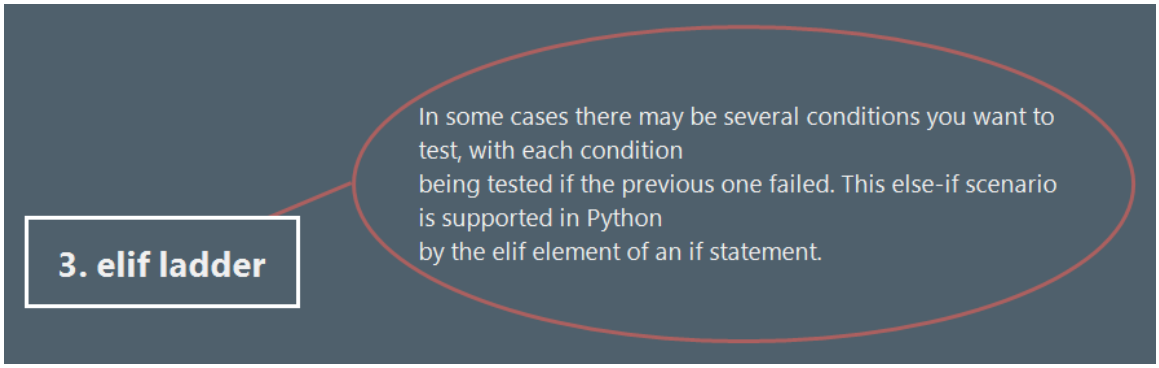
Example:



For example:

```
num = int(input('Enter yet another number: '))
if num < 0:
print('Its negative')
else:
print('Its not negative')
```

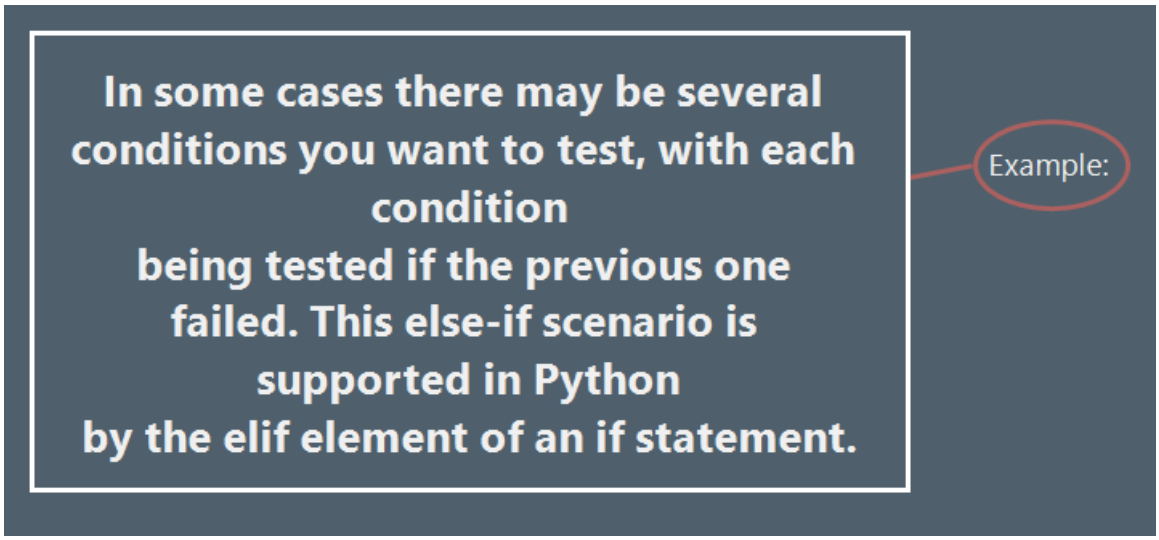
3. elif ladder



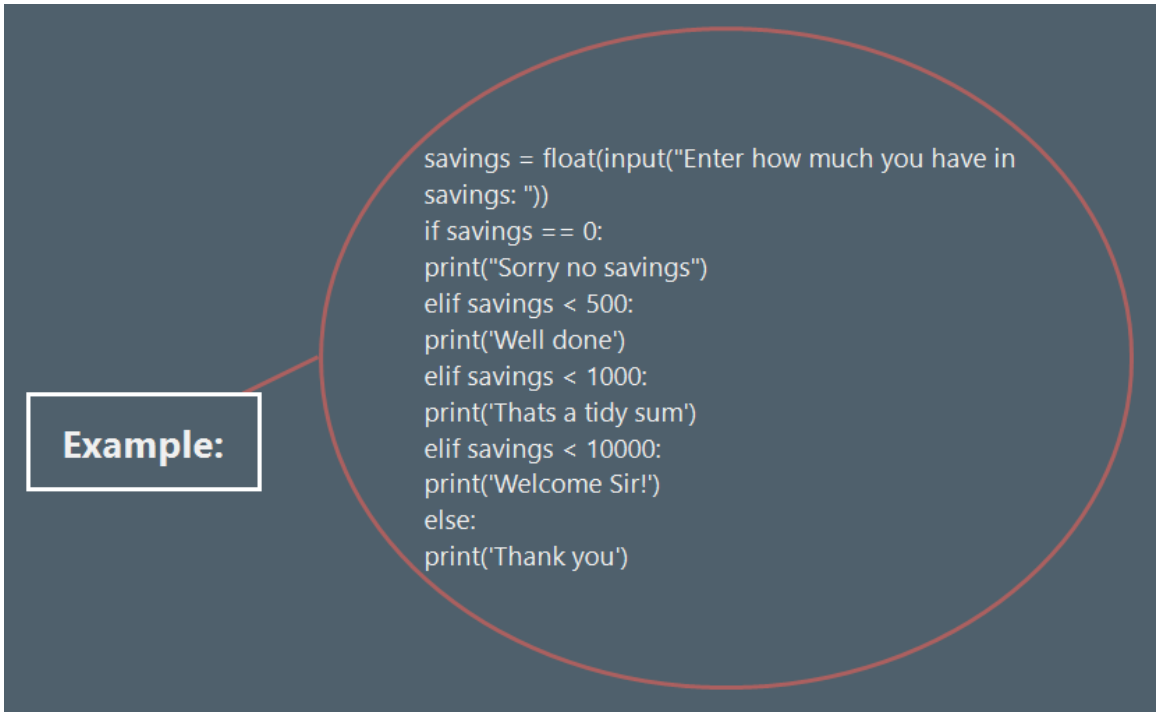
In some cases there may be several conditions you want to test, with each condition

being tested if the previous one failed. This else-if scenario is supported in Python

by the elif element of an if statement.

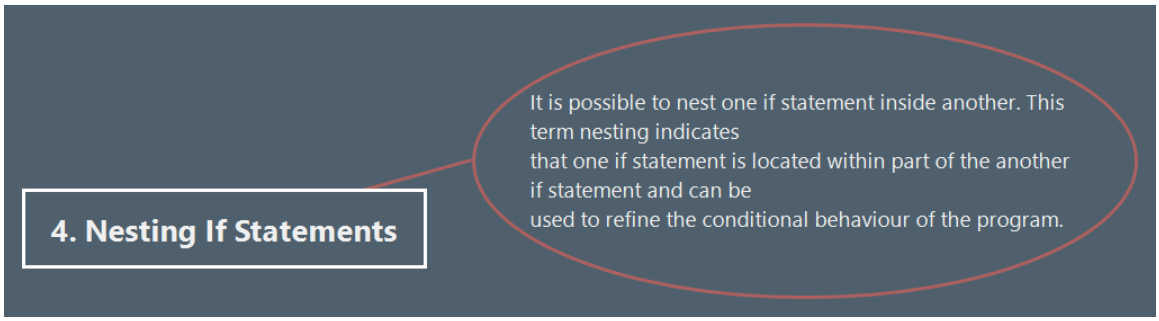


Example:




```
savings = float(input("Enter how much you have in savings: "))
if savings == 0:
print("Sorry no savings")
elif savings < 500:
print('Well done')
elif savings < 1000:
print('Thats a tidy sum')
elif savings < 10000:
print('Welcome Sir!')
else:
print('Thank you')
```

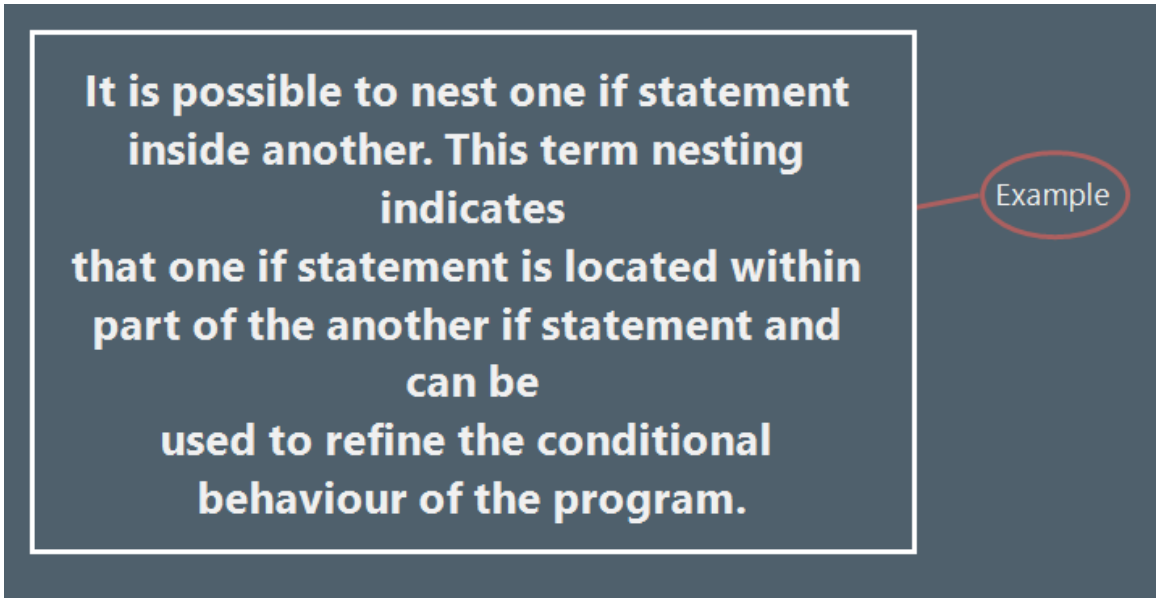
4. Nesting If Statements



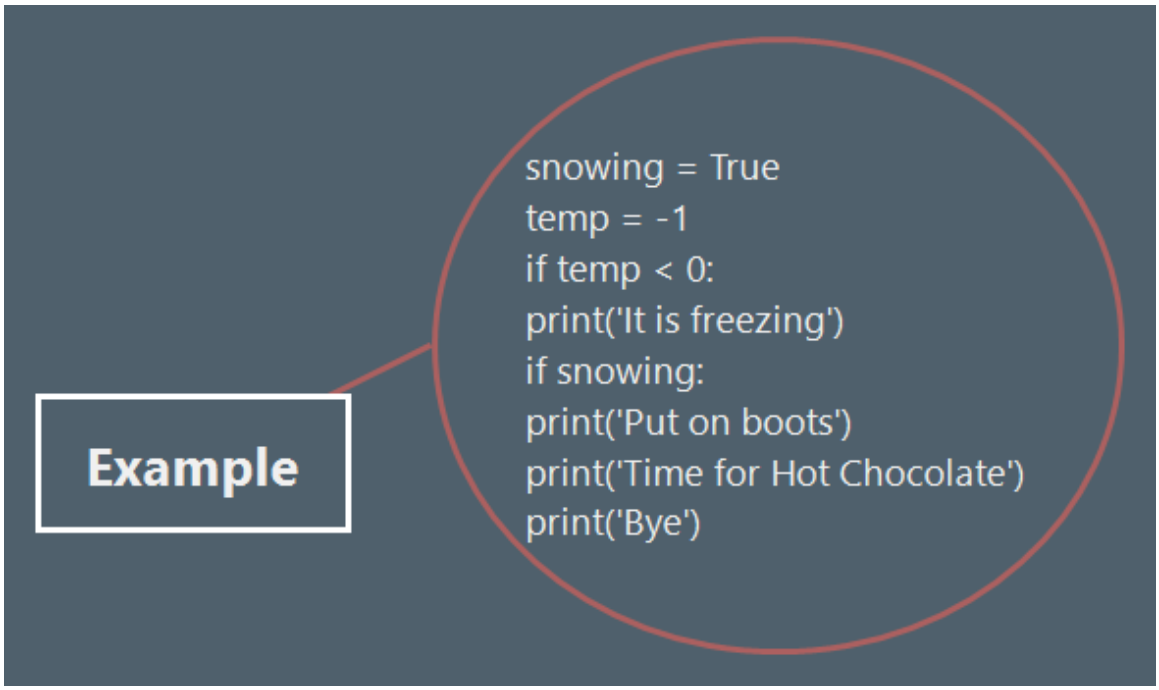
It is possible to nest one if statement inside another. This term nesting indicates

that one if statement is located within part of the another if statement and can be

used to refine the conditional behaviour of the program.

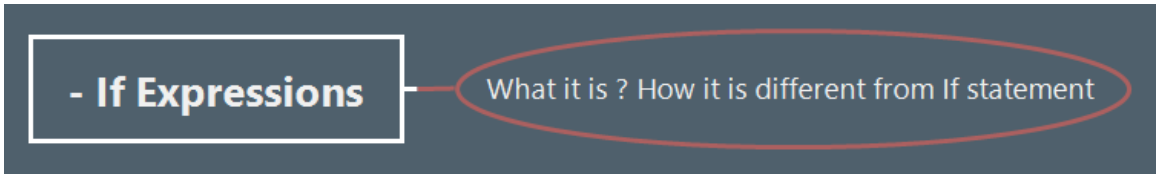


Example




```
snowing = True
temp = -1
if temp < 0:
print('It is freezing')
if snowing:
print('Put on boots')
print('Time for Hot Chocolate')
print('Bye')
```

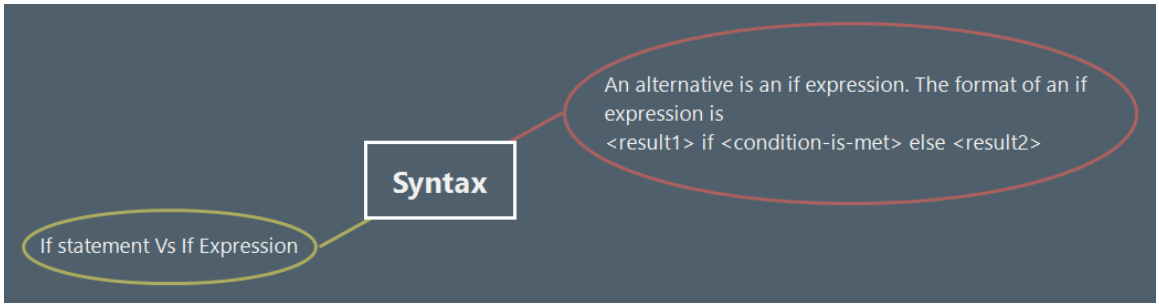
3.2.4. - If Expressions



What it is ? How it is different from If statement

An if expression is a short hand form of an if statement that returns a value. In fact, the difference between an expression and a statement in a programming language
is just that; expressions return a value; statements do not.

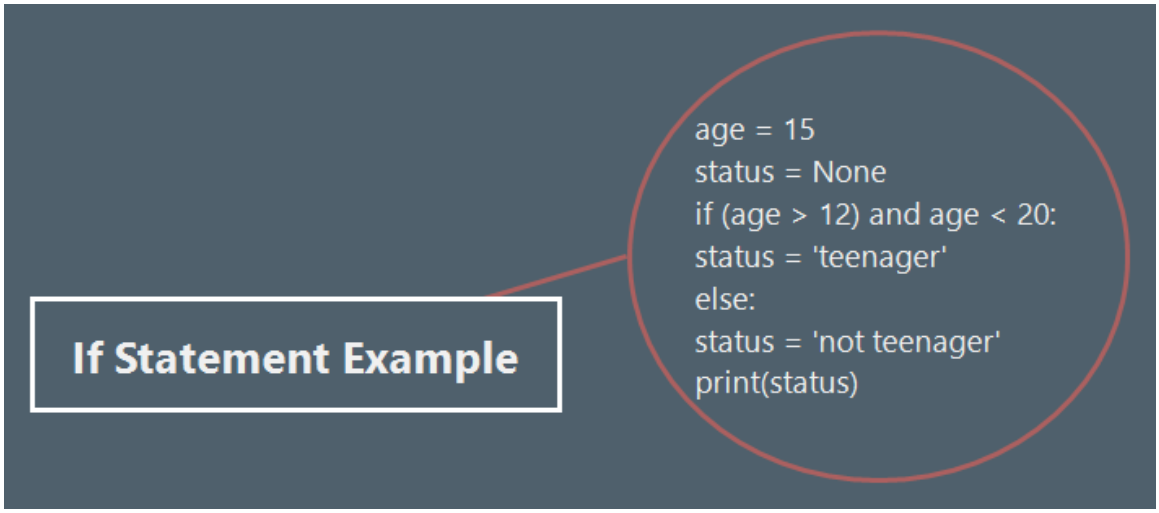
Syntax



An alternative is an if expression. The format of an if expression is <result1> if <condition-is-met> else <result2>

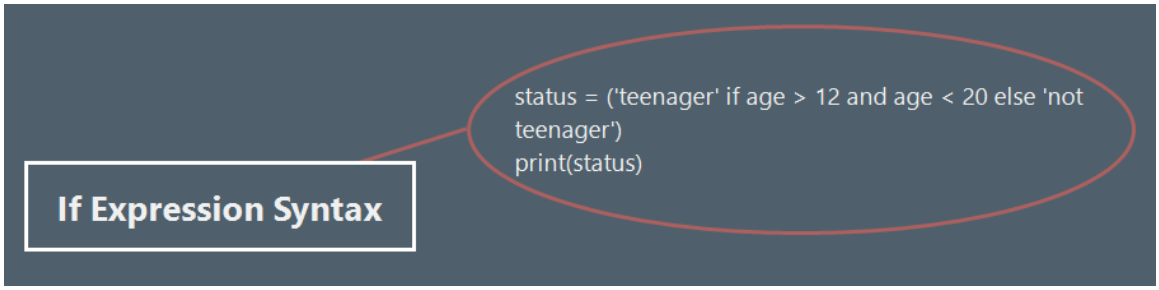
If statement Vs If Expression

If Statement Example



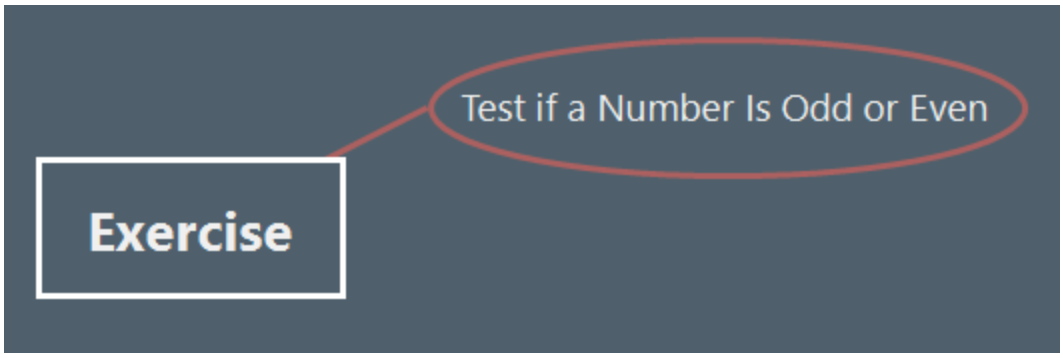

```
age = 15
status = None
if (age > 12) and age < 20:
status = 'teenager'
else:
status = 'not teenager'
print(status)
```

If Expression Syntax

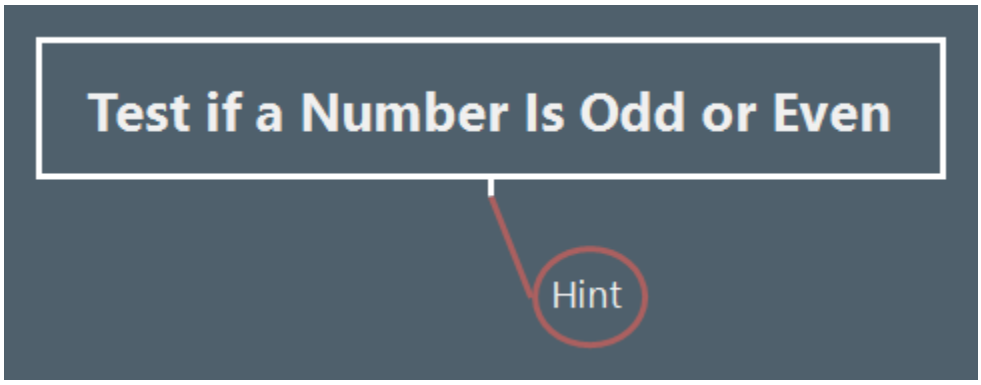



```
status = ('teenager' if age > 12 and age < 20 else 'not
teenager')
print(status)
```

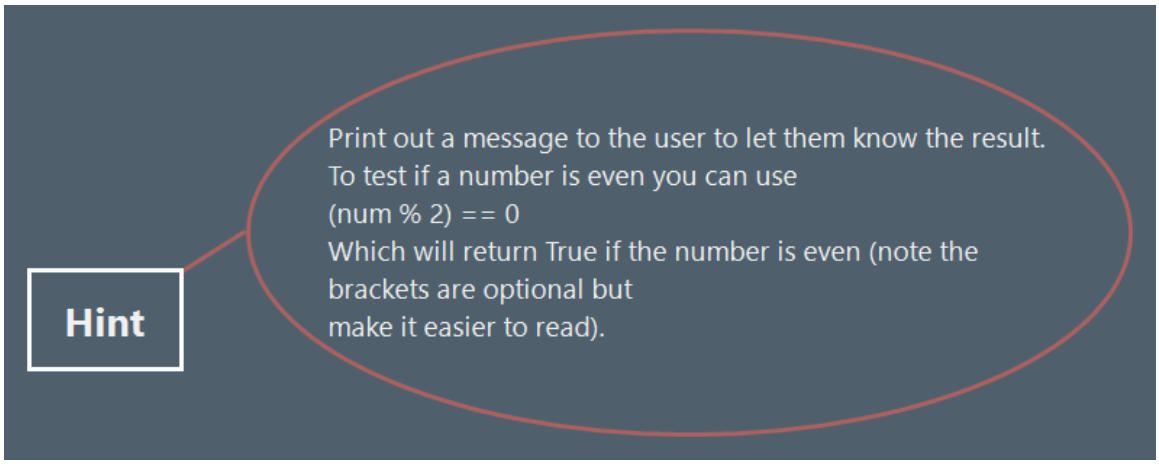
3.3. Exercise



3.3.1. Test if a Number Is Odd or Even



Hint



Print out a message to the user to let them know the result.

To test if a number is even you can use

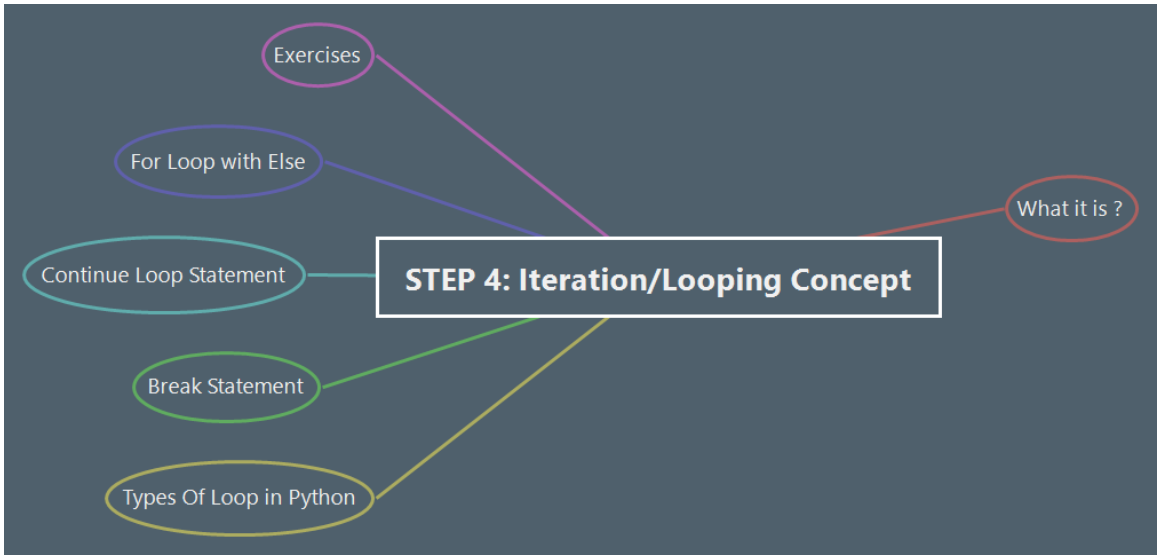
(num % 2) == 0

Which will return True if the number is even (note the brackets are optional

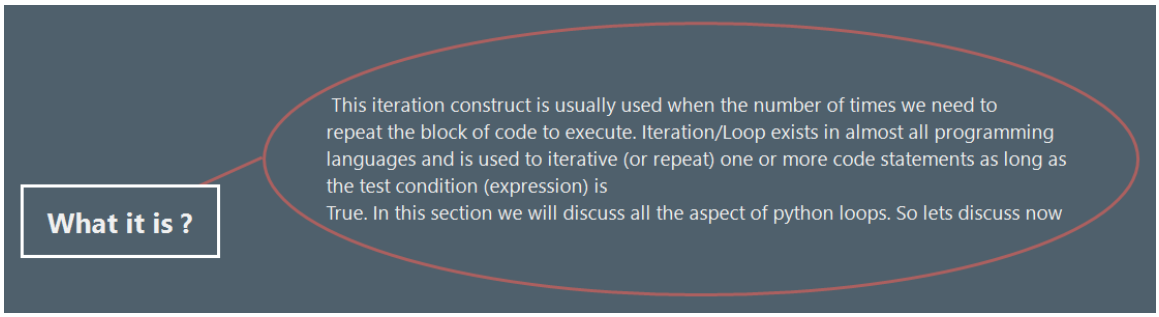
but

make it easier to read).

4. STEP 4: Iteration/Looping Concept



4.1. What it is ?

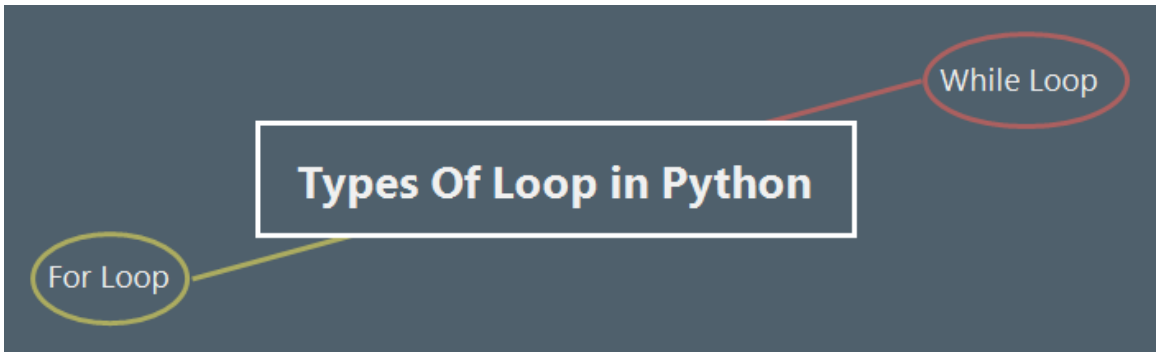


4.1.1. This iteration construct is usually used when the number of times we need to

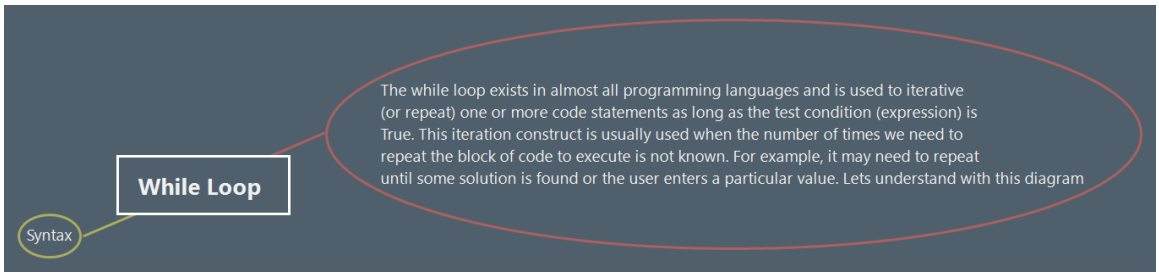
repeat the block of code to execute. Iteration/Loop exists in almost all programming languages and is used to iterative (or repeat) one or more code statements as long as the test condition (expression) is

True. In this section we will discuss all the aspect of python loops. So lets discuss now

4.2. Types Of Loop in Python

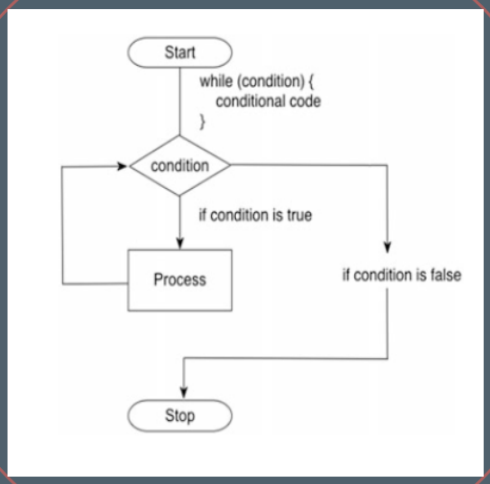


4.2.1. While Loop

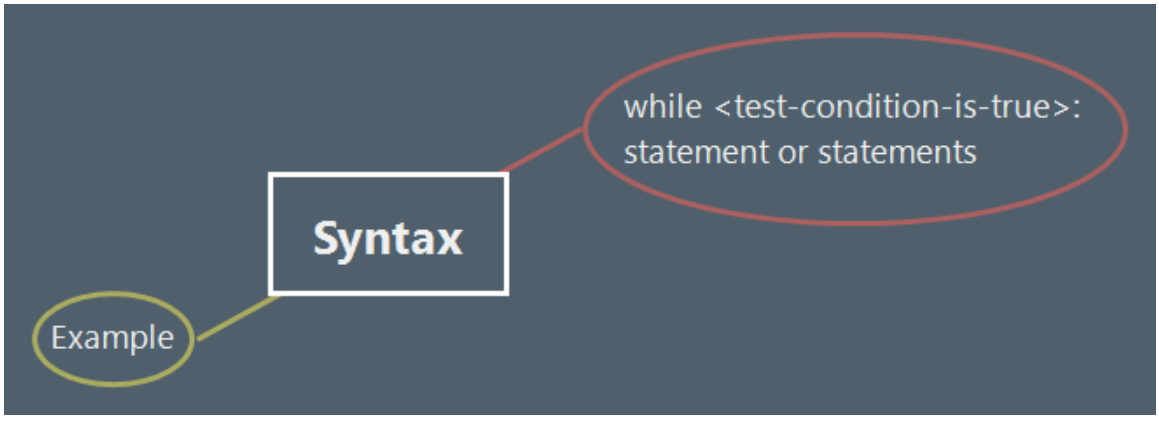


The while loop exists in almost all programming languages and is used to iterative
(or repeat) one or more code statements as long as the test condition (expression) is
True. This iteration construct is usually used when the number of times we need to
repeat the block of code to execute is not known. For example, it may need to repeat
until some solution is found or the user enters a particular value. Lets understand with this diagram

The while loop exists in almost all programming languages and is used to iterative
(or repeat) one or more code statements as long as the test condition (expression) is
True. This iteration construct is usually used when the number of times we need to
repeat the block of code to execute is not known. For example, it may need to repeat
until some solution is found or the user enters a particular value.
Lets understand with this diagram



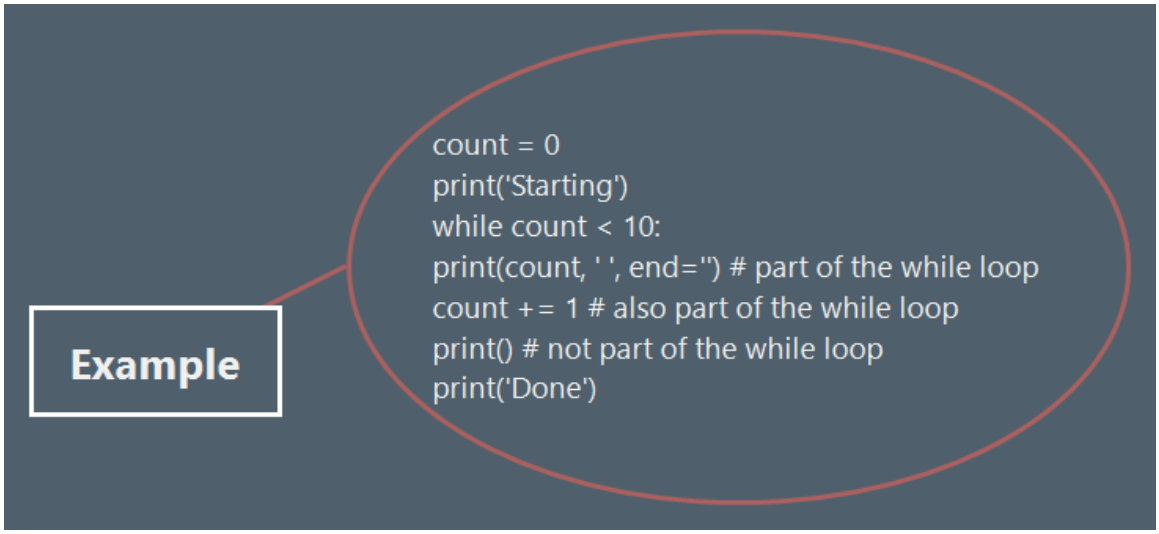
Syntax



while <test-condition-is-true>:

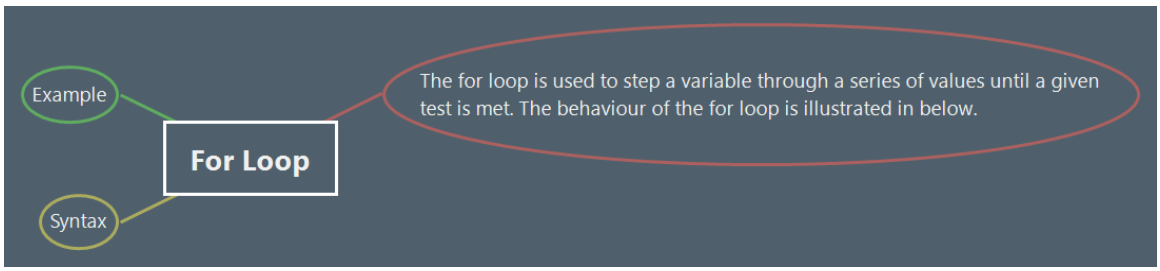
statement or statements

Example



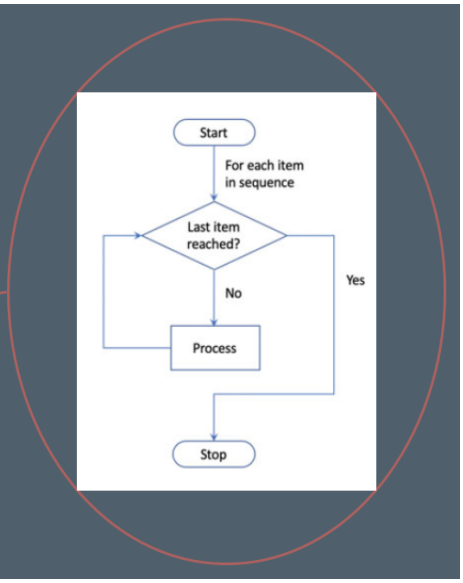

```
count = 0
print('Starting')
while count < 10:
print(count, ' ', end='') # part of the while loop
count += 1 # also part of the while loop
print() # not part of the while loop
print('Done')
```

4.2.2. For Loop

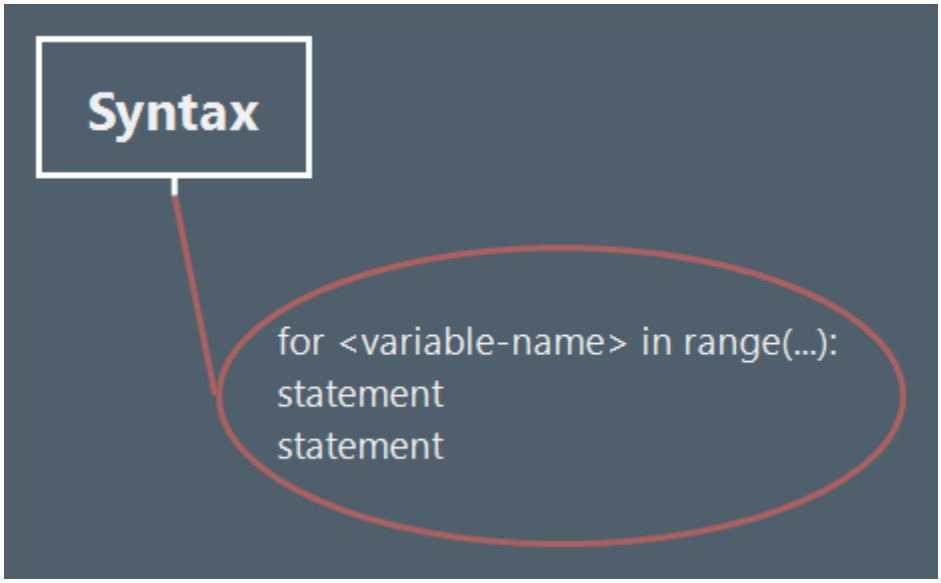


The for loop is used to step a variable through a series of values until a given test is met. The behaviour of the for loop is illustrated in below.

The for loop is used to step a variable through a series of values until a given
test is met. The behaviour of the for loop is illustrated in below.



Syntax

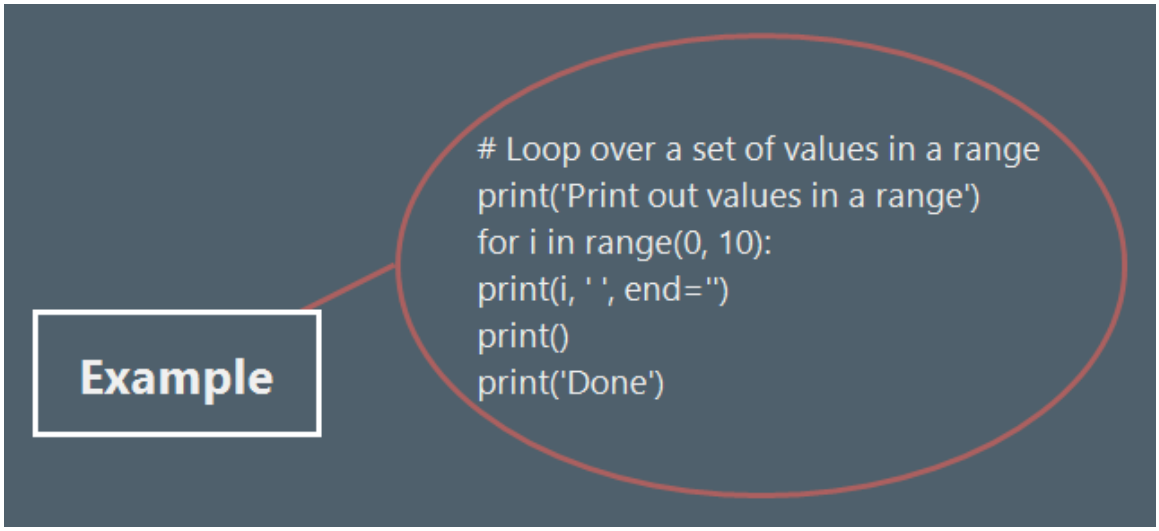


for <variable-name> in range(...):

statement

statement

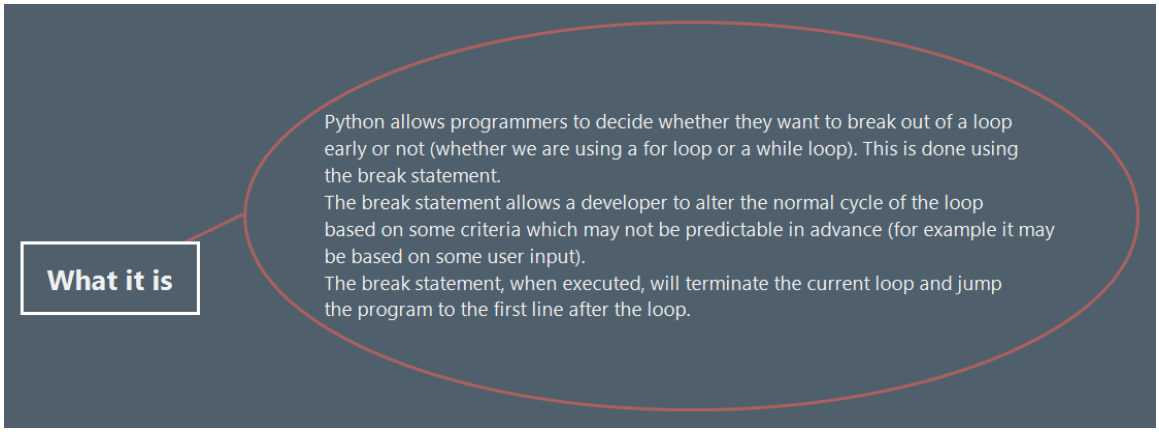
Example




```
# Loop over a set of values in a range
print('Print out values in a range')
for i in range(0, 10):
print(i, ' ', end='')
print()
print('Done')
```

4.3. Break Statement

4.3.1. What it is

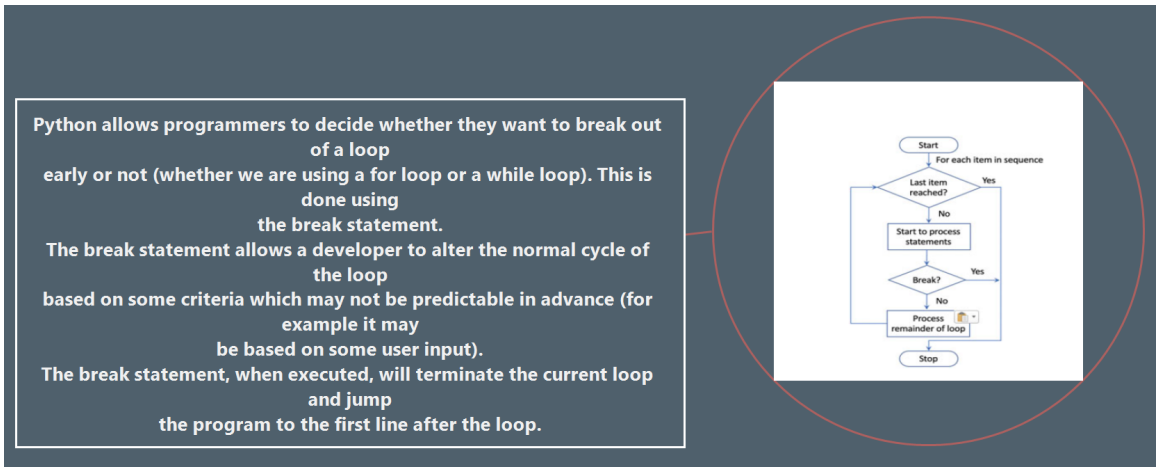


Python allows programmers to decide whether they want to break out of a loop early or not (whether we are using a for loop or a while loop). This is done using the break statement.

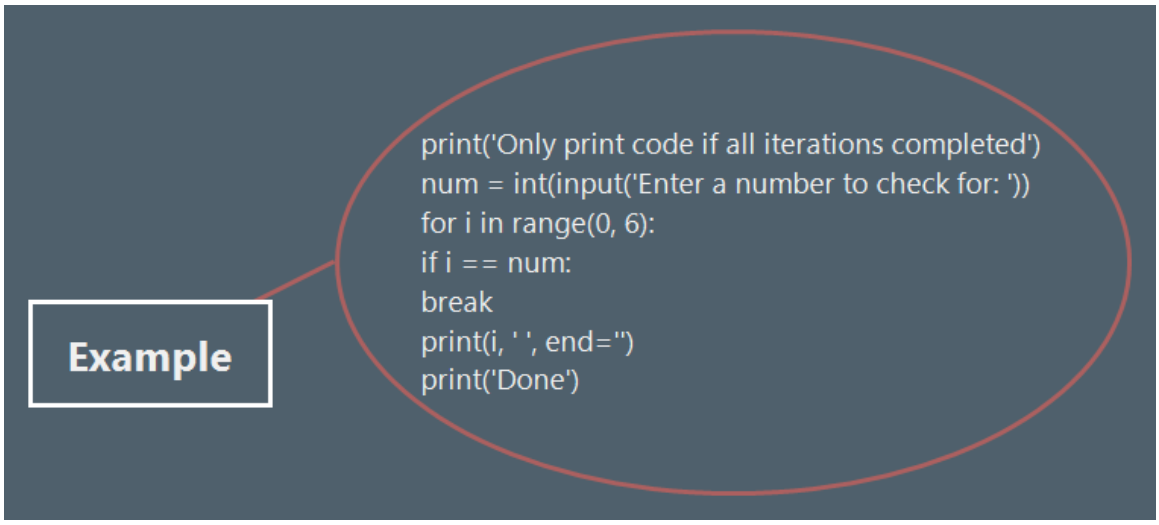
The break statement allows a developer to alter the normal cycle of the loop based on some criteria which may not be predictable in advance (for example it may

be based on some user input).

The break statement, when executed, will terminate the current loop and jump the program to the first line after the loop.

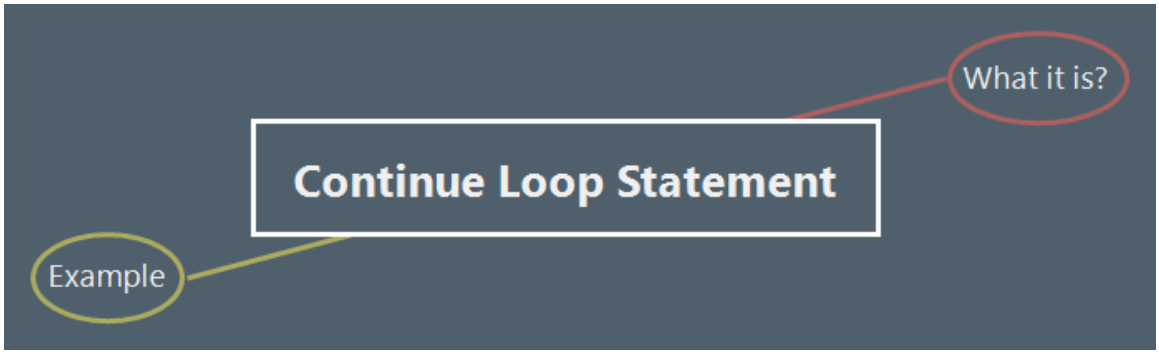


4.3.2. Example

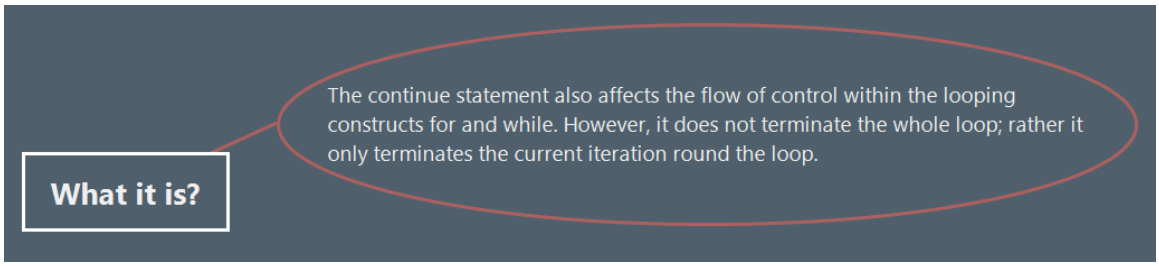



```
print('Only print code if all iterations completed')
num = int(input('Enter a number to check for: '))
for i in range(0, 6):
if i == num:
break
print(i, ' ', end='')
print('Done')
```

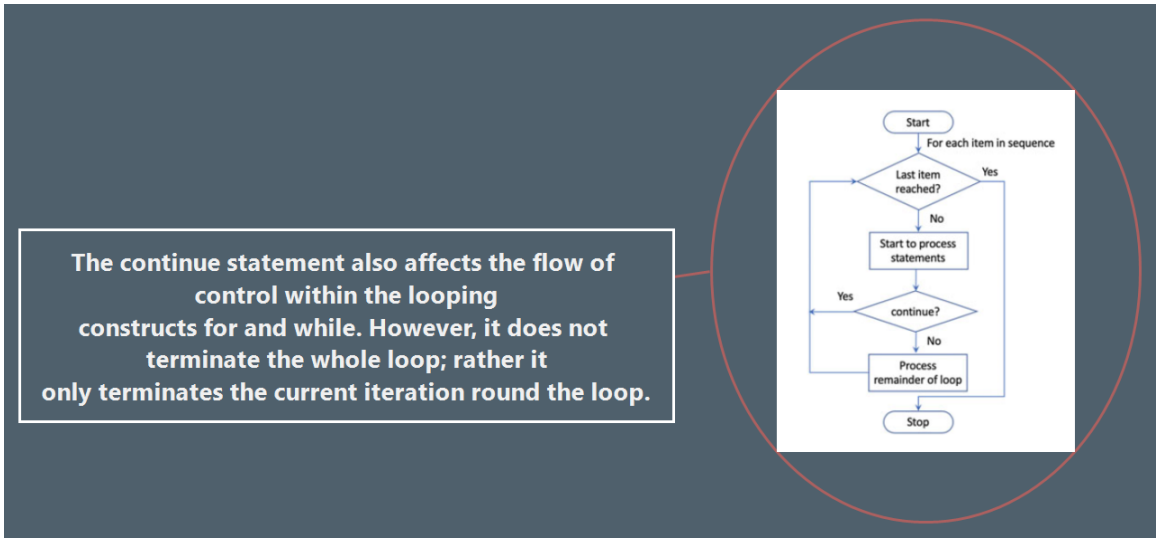
4.4. Continue Loop Statement



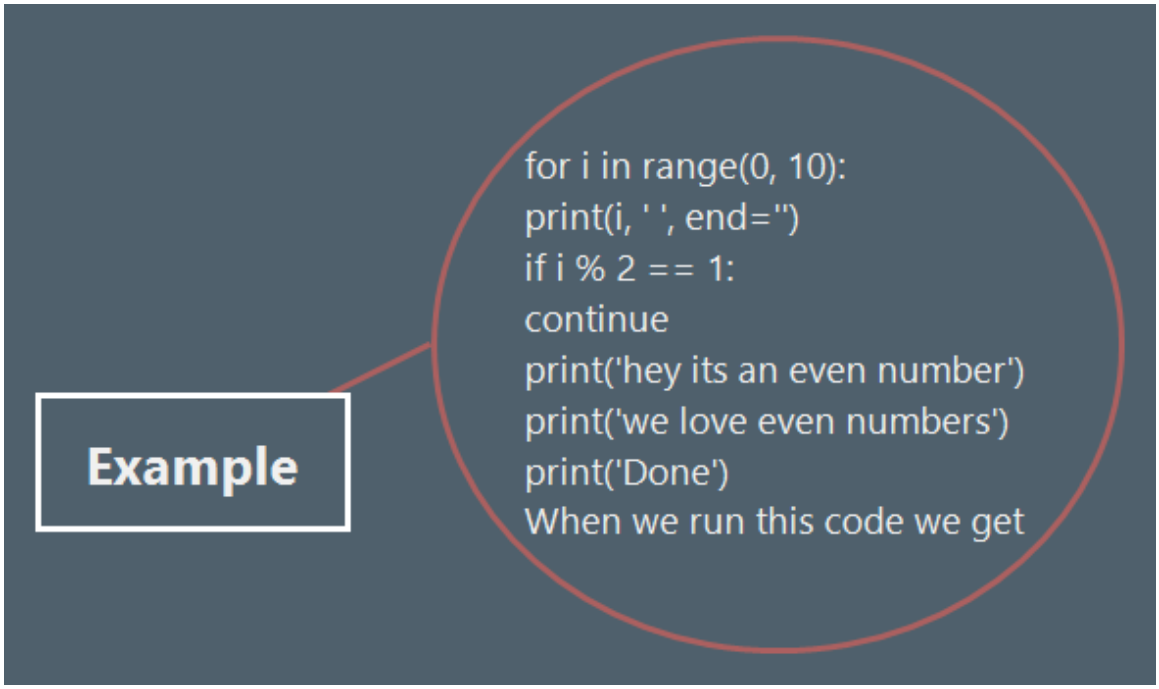
4.4.1. What it is?



The continue statement also affects the flow of control within the looping constructs for and while. However, it does not terminate the whole loop; rather it
only terminates the current iteration round the loop.



4.4.2. Example



for i in range(0, 10):

print(i, ' ', end='')

if i % 2 == 1:

continue

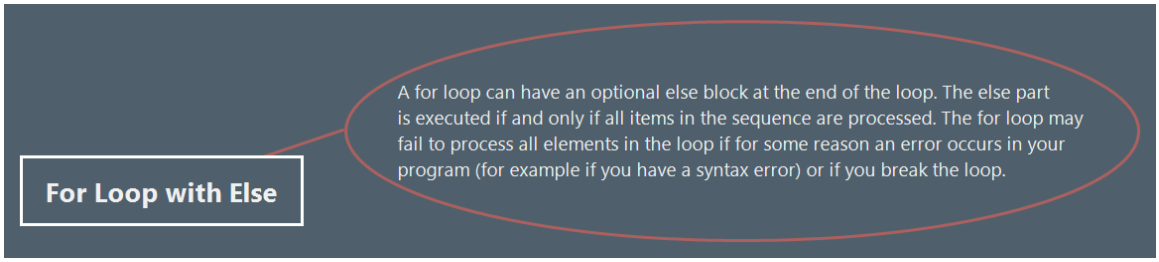
print('hey its an even number')

print('we love even numbers')

print('Done')

When we run this code we get

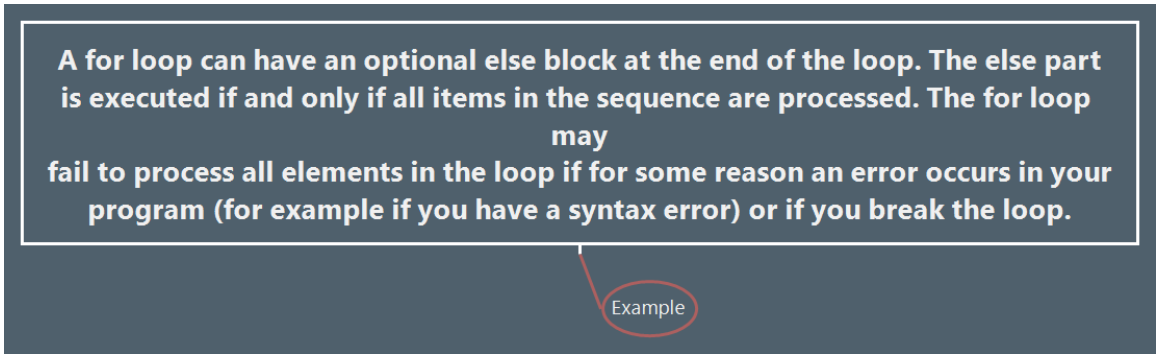
4.5. For Loop with Else



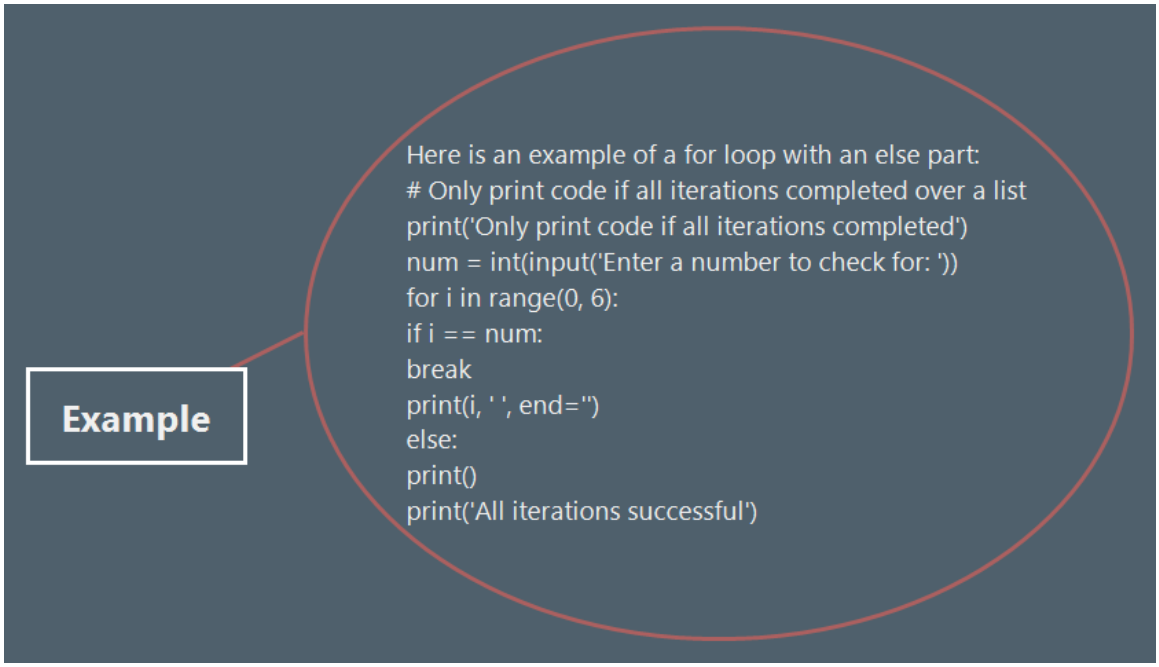
4.5.1. A for loop can have an optional else block at the end of the loop. The else part

is executed if and only if all items in the sequence are processed. The for loop may

fail to process all elements in the loop if for some reason an error occurs in your program (for example if you have a syntax error) or if you break the loop.



Example



Here is an example of a for loop with an else part:

 # Only print code if all iterations completed over a list

print('Only print code if all iterations completed')

num = int(input('Enter a number to check for: '))

for i in range(0, 6):

if i == num:

break

print(i, ' ', end='')

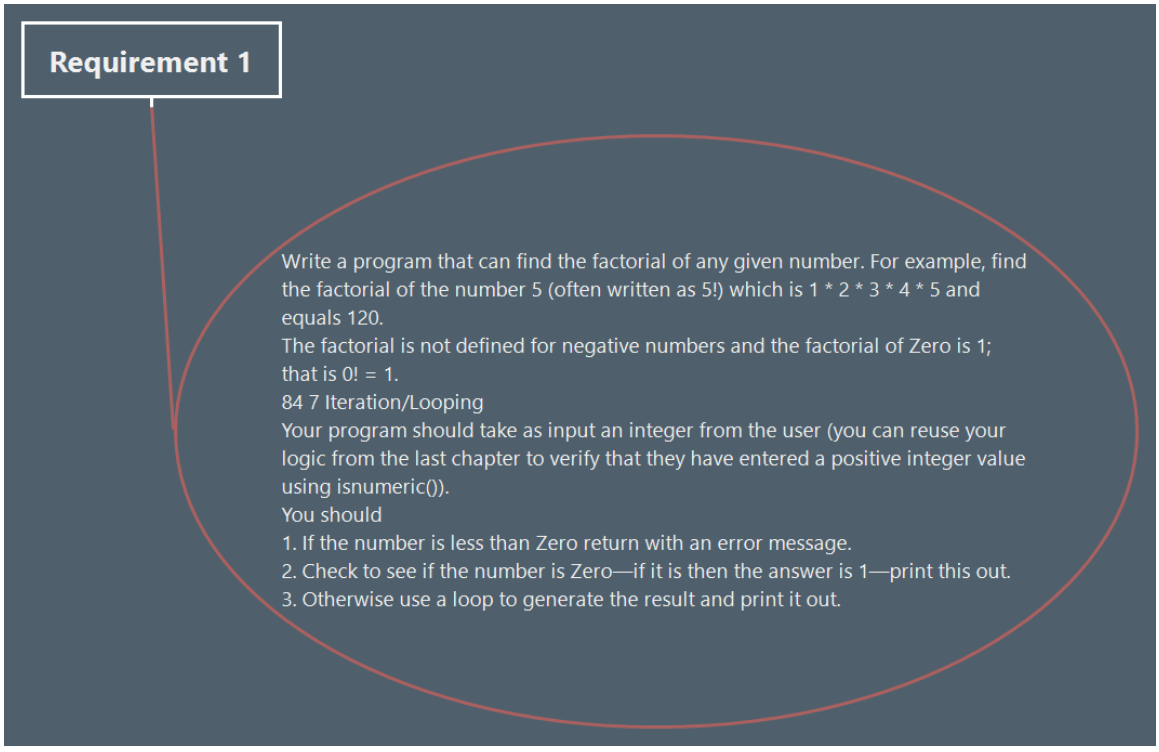
else:

print()

print('All iterations successful')

4.6. Exercises

4.6.1. Requirement 1



Write a program that can find the factorial of any given number. For example, find

the factorial of the number 5 (often written as 5!) which is 1 * 2 * 3 * 4 * 5 and equals 120.

The factorial is not defined for negative numbers and the factorial of Zero is 1; that is 0! = 1.

84 7 Iteration/Looping

Your program should take as input an integer from the user (you can reuse your logic from the last chapter to verify that they have entered a positive integer value

using isnumeric()).

You should

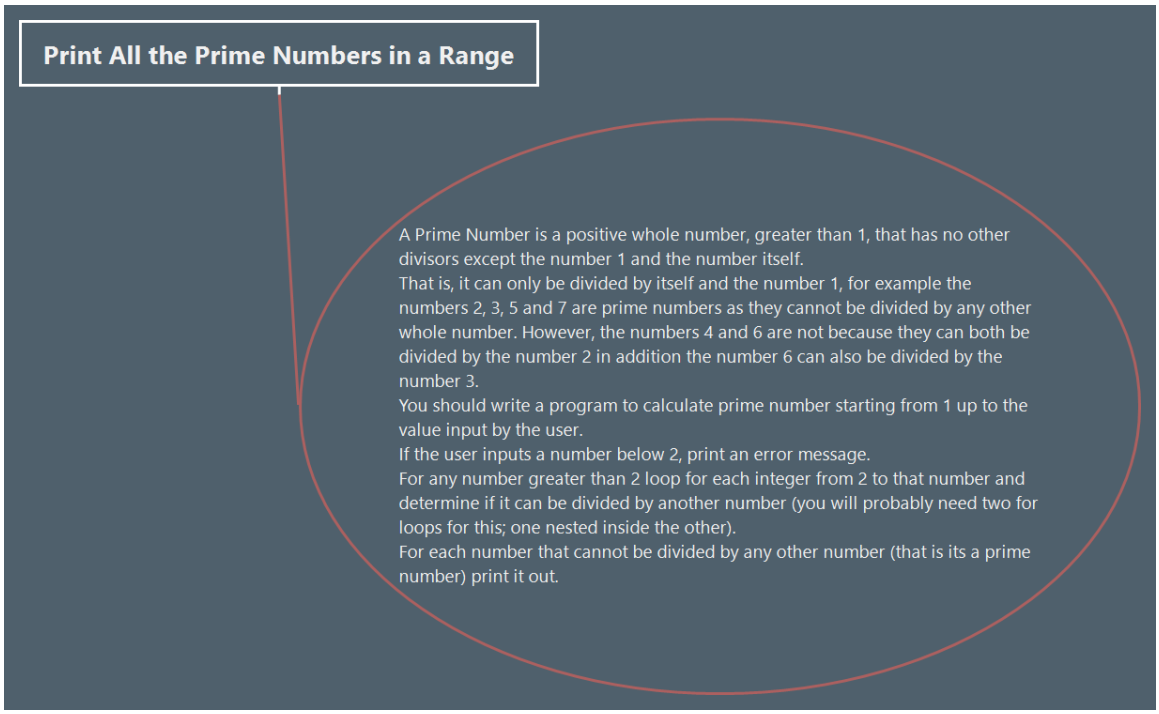
1. If the number is less than Zero return with an error message.

2. Check to see if the number is Zero—if it is then the answer is 1—print this out.

3. Otherwise use a loop to generate the result and print it out.

4.6.2. Requirement 2

Print All the Prime Numbers in a Range



A Prime Number is a positive whole number, greater than 1, that has no other divisors except the number 1 and the number itself.
That is, it can only be divided by itself and the number 1, for example the numbers 2, 3, 5 and 7 are prime numbers as they cannot be divided by any other
whole number. However, the numbers 4 and 6 are not because they can both be
divided by the number 2 in addition the number 6 can also be divided by the number 3.
You should write a program to calculate prime number starting from 1 up to the
value input by the user.
If the user inputs a number below 2, print an error message.
For any number greater than 2 loop for each integer from 2 to that number and
determine if it can be divided by another number (you will probably need two for

loops for this; one nested inside the other).

For each number that cannot be divided by any other number (that is its a prime

number) print it out.

5. STEP 5: Tools To Convert From Requirement To Program

5.1. Functional Decomposition

5.1.1. Functional Decomposition is one way in which a system can be broken down into

its constituent parts. For example, for a computer payroll system to calculate how much an hourly paid employee should receive it might be necessary to:

1. Load the employee's details from some form of permanent storage (such as a file

or a database).

2. Load how many hours the employee has worked for that week (possibly from another system that records the number of hours worked).

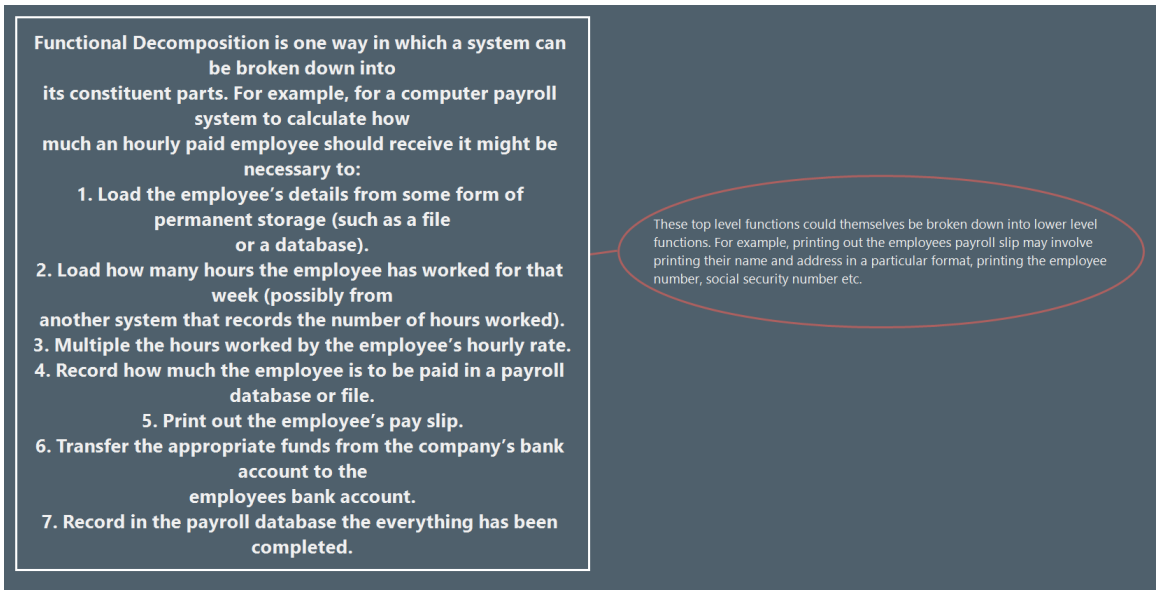
3. Multiple the hours worked by the employee's hourly rate.

4. Record how much the employee is to be paid in a payroll database or file.

5. Print out the employee's pay slip.

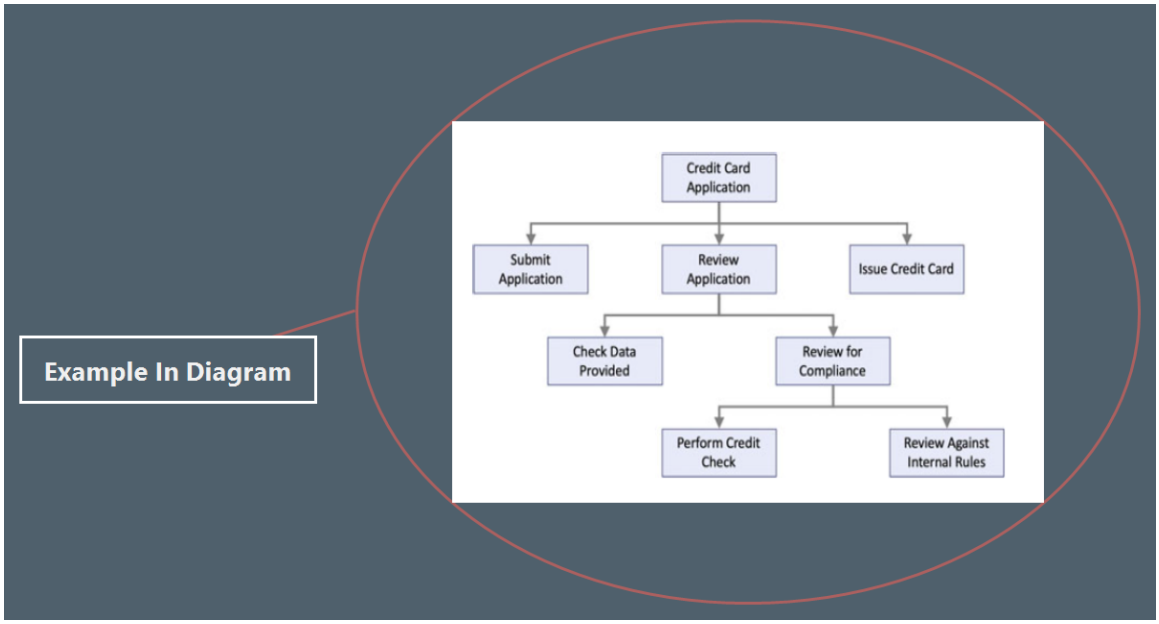
6. Transfer the appropriate funds from the company's bank account to the employees bank account.

7. Record in the payroll database the everything has been completed.

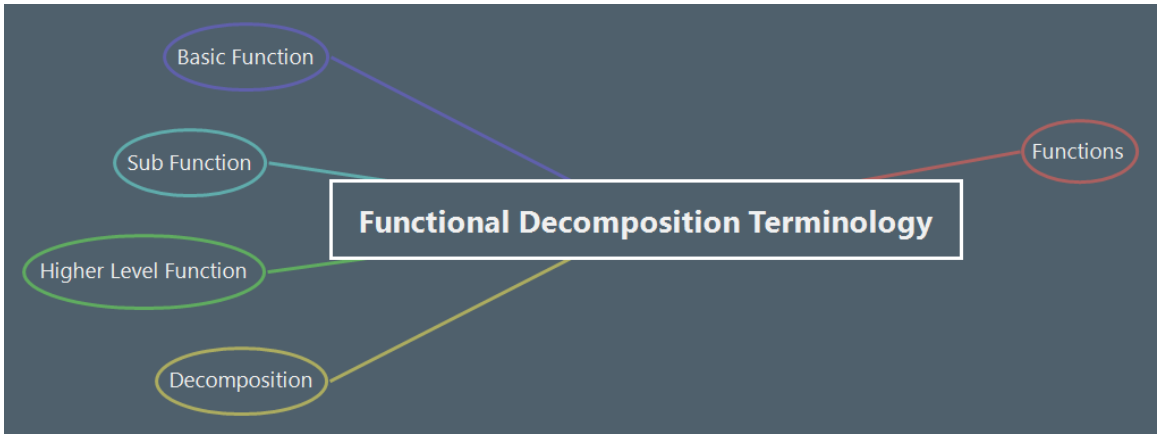


These top level functions could themselves be broken down into lower level functions. For example, printing out the employees payroll slip may involve printing their name and address in a particular format, printing the employee number, social security number etc.

5.1.2. Example In Diagram



5.1.3. Functional Decomposition Terminology



Functions

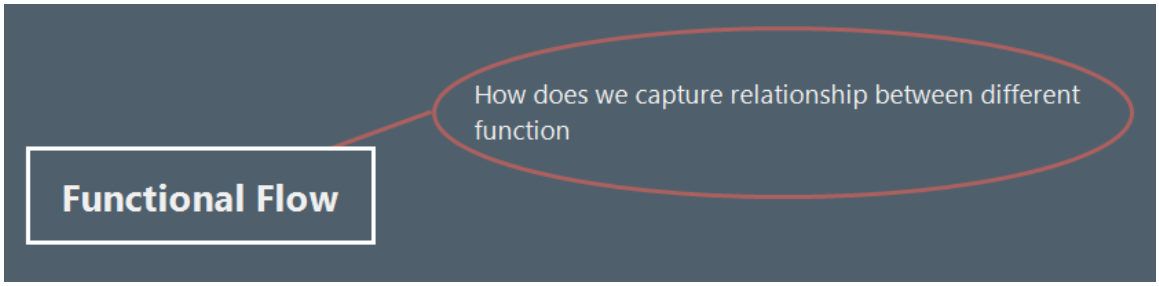
Decomposition

Higher Level Function

Sub Function

Basic Function

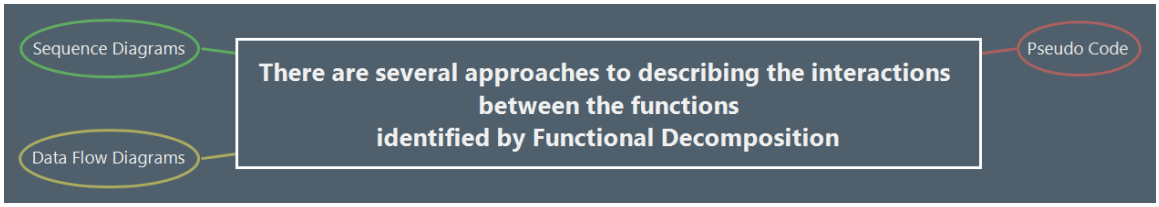
5.2. Functional Flow



5.2.1. How does we capture relationship between different function

There are several approaches to describing the interactions between the functions

identified by Functional Decomposition

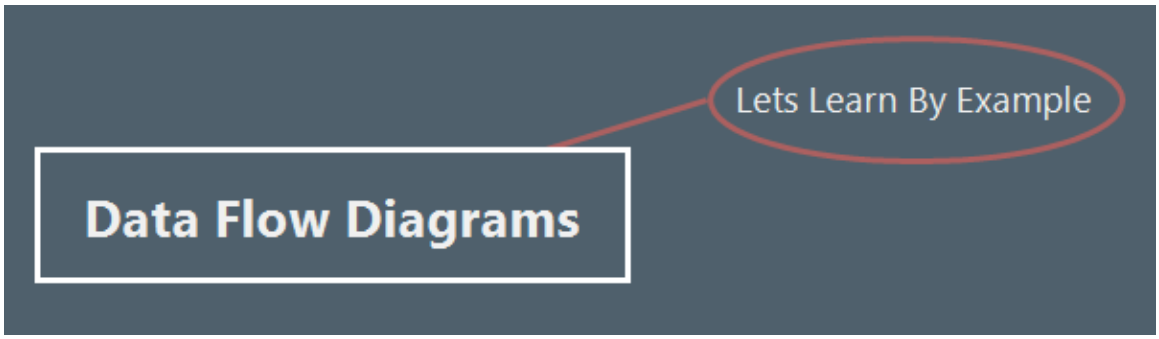


Pseudo Code

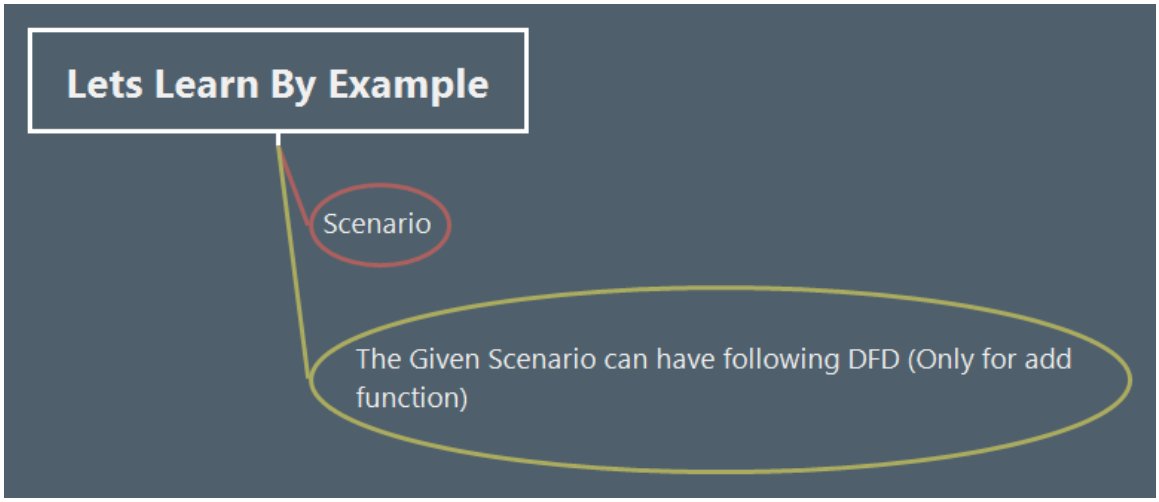
Data Flow Diagrams

Sequence Diagrams

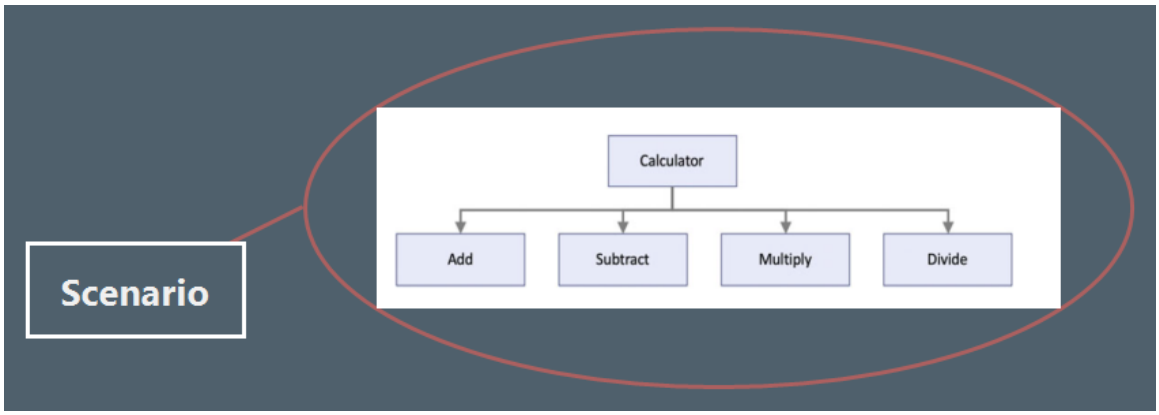
5.3. Data Flow Diagrams



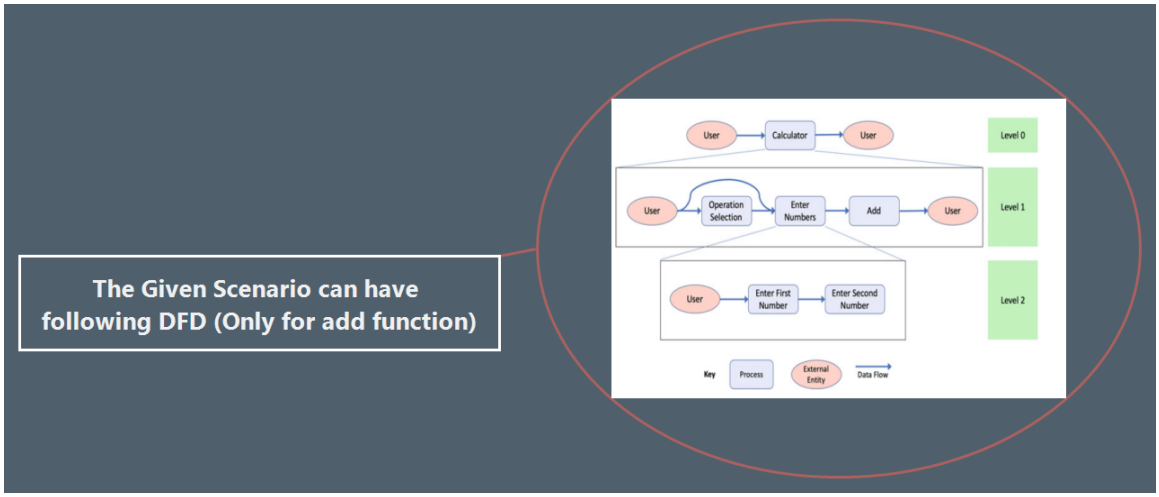
5.3.1. Lets Learn By Example



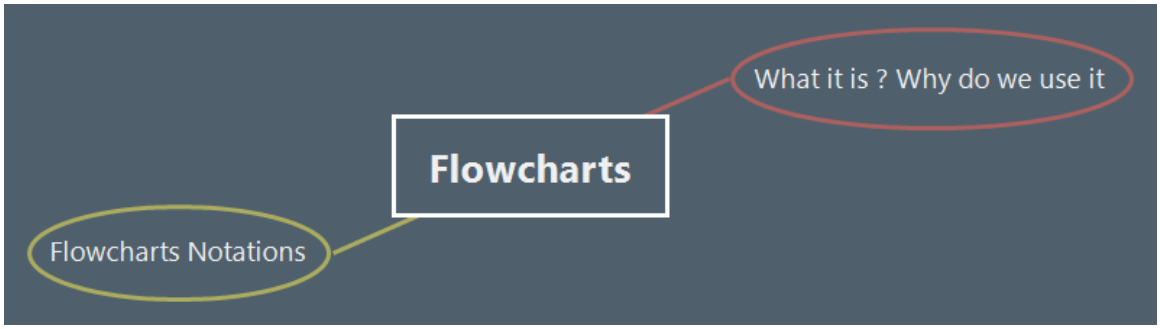
Scenario



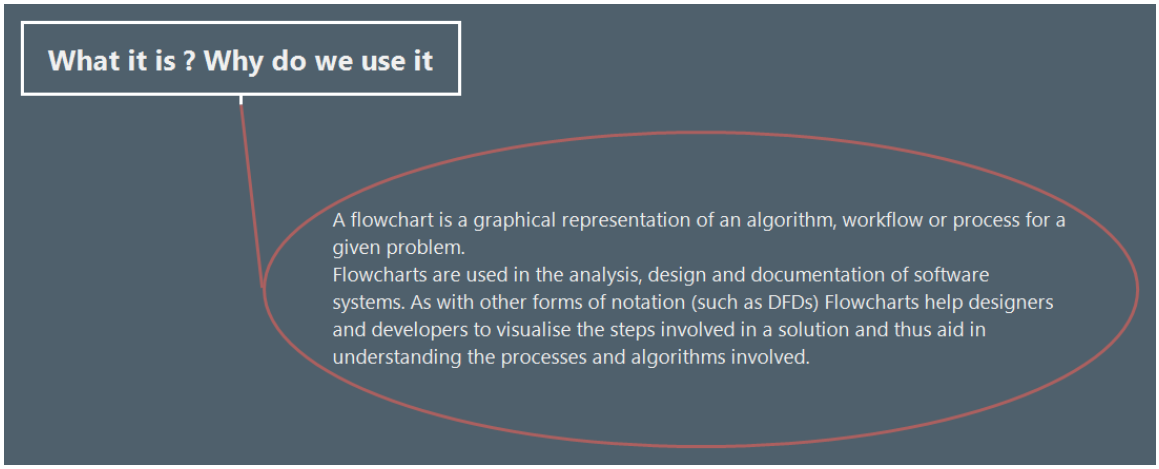
The Given Scenario can have following DFD (Only for add function)



5.4. Flowcharts



5.4.1. What it is ? Why do we use it



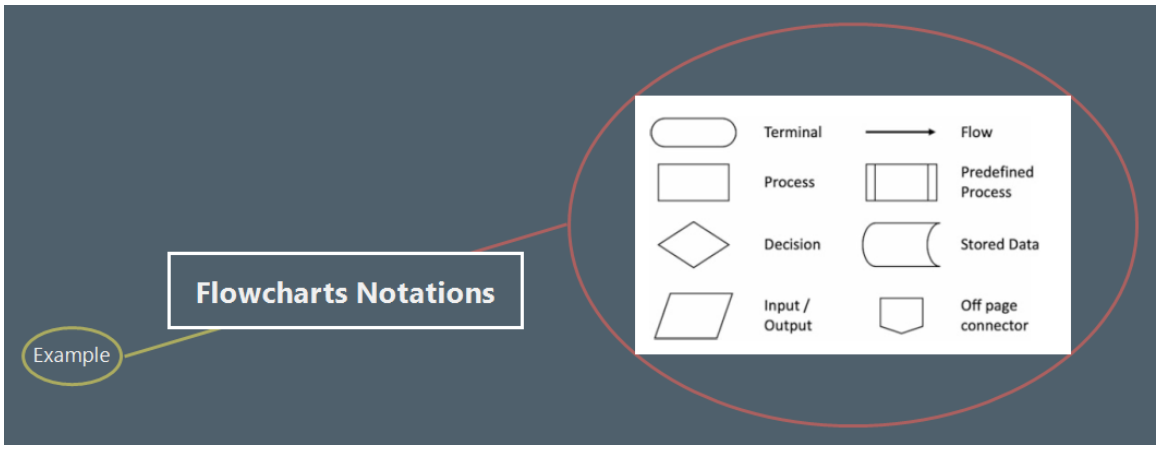
A flowchart is a graphical representation of an algorithm, workflow or process for a

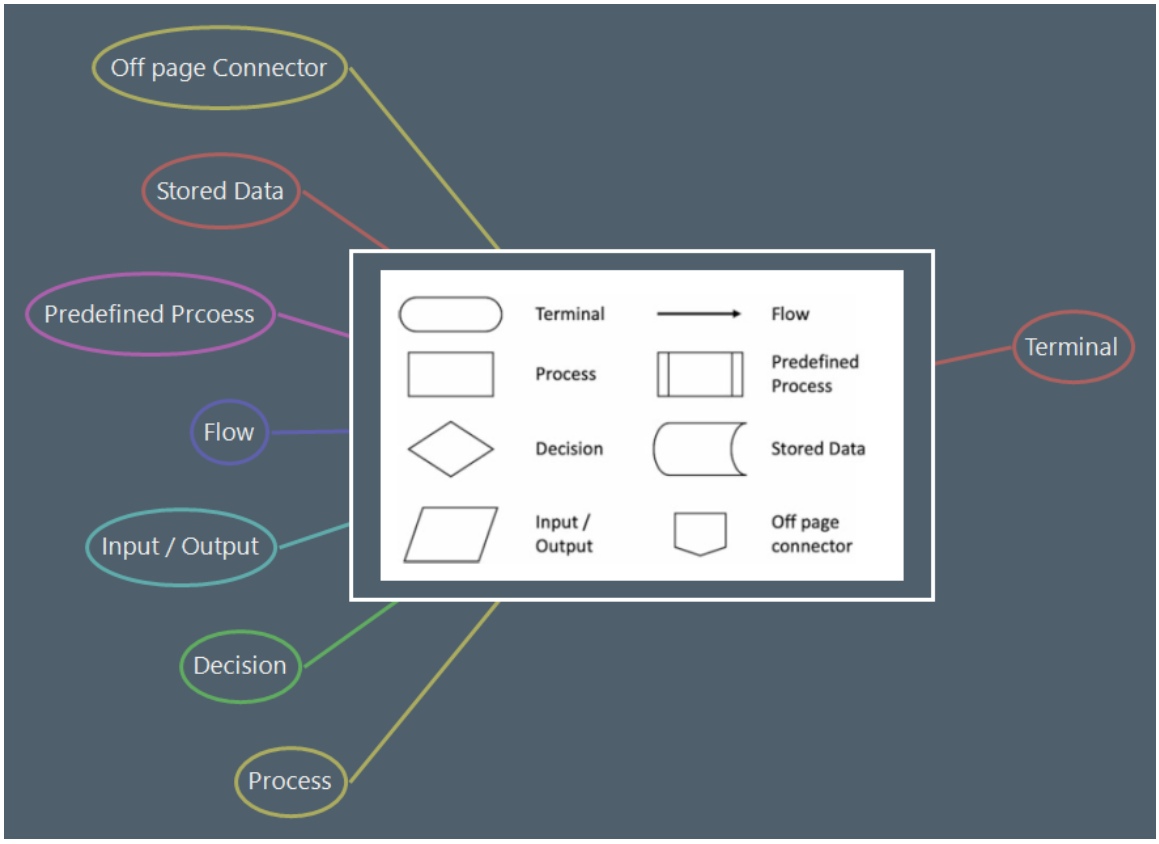
given problem.

Flowcharts are used in the analysis, design and documentation of software systems. As with other forms of notation (such as DFDs) Flowcharts help designers

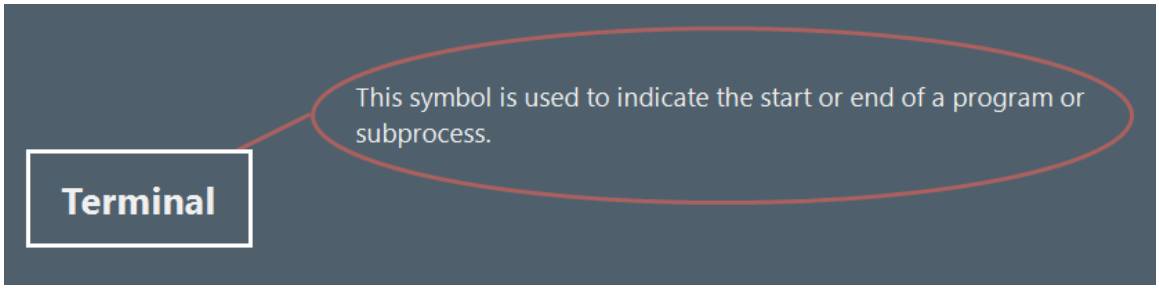
and developers to visualise the steps involved in a solution and thus aid in understanding the processes and algorithms involved.

5.4.2. Flowcharts Notations



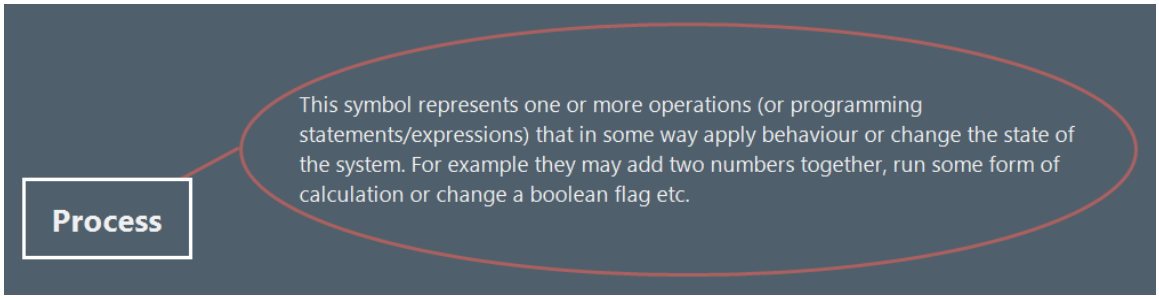


Terminal



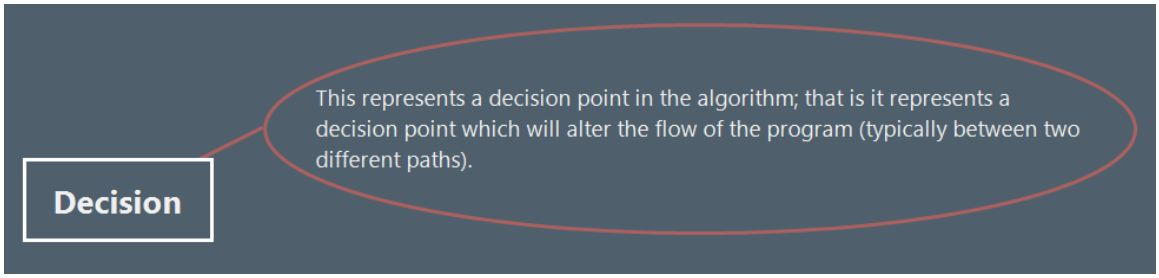
This symbol is used to indicate the start or end of a program or subprocess.

Process



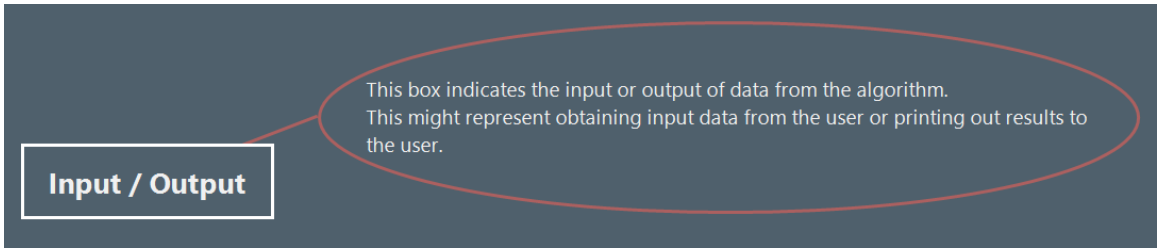
This symbol represents one or more operations (or programming statements/expressions) that in some way apply behaviour or change the state of
the system. For example they may add two numbers together, run some form of
calculation or change a boolean flag etc.

Decision



This represents a decision point in the algorithm; that is it represents a decision point which will alter the flow of the program (typically between two
different paths).

Input / Output

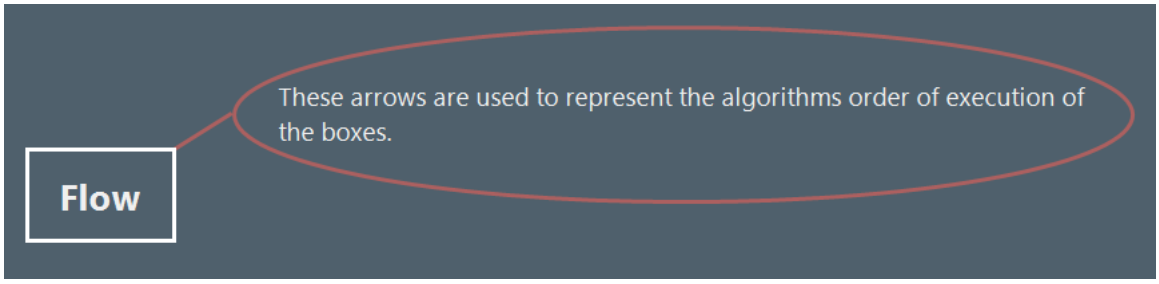


This box indicates the input or output of data from the algorithm.

This might represent obtaining input data from the user or printing out results to

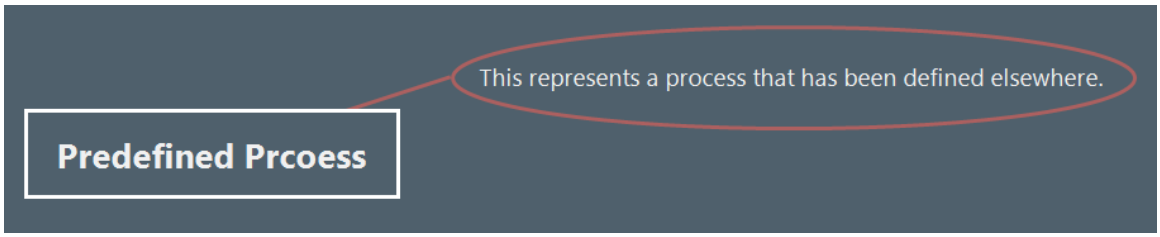
the user.

Flow



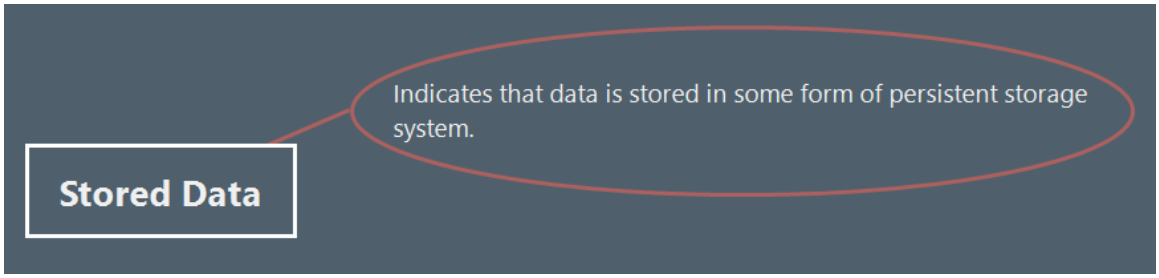
These arrows are used to represent the algorithms order of execution of the boxes.

Predefined Prcoess



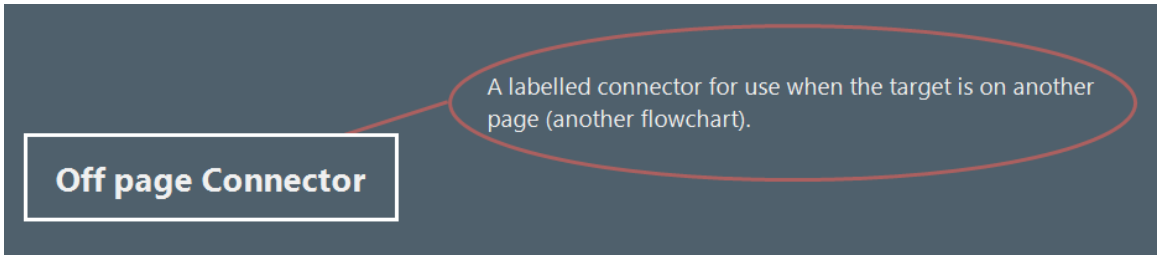
This represents a process that has been defined elsewhere.

Stored Data



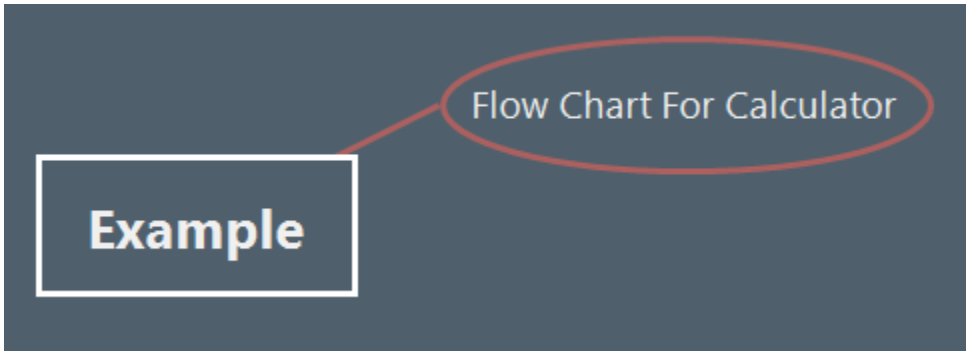
Indicates that data is stored in some form of persistent storage system.

Off page Connector

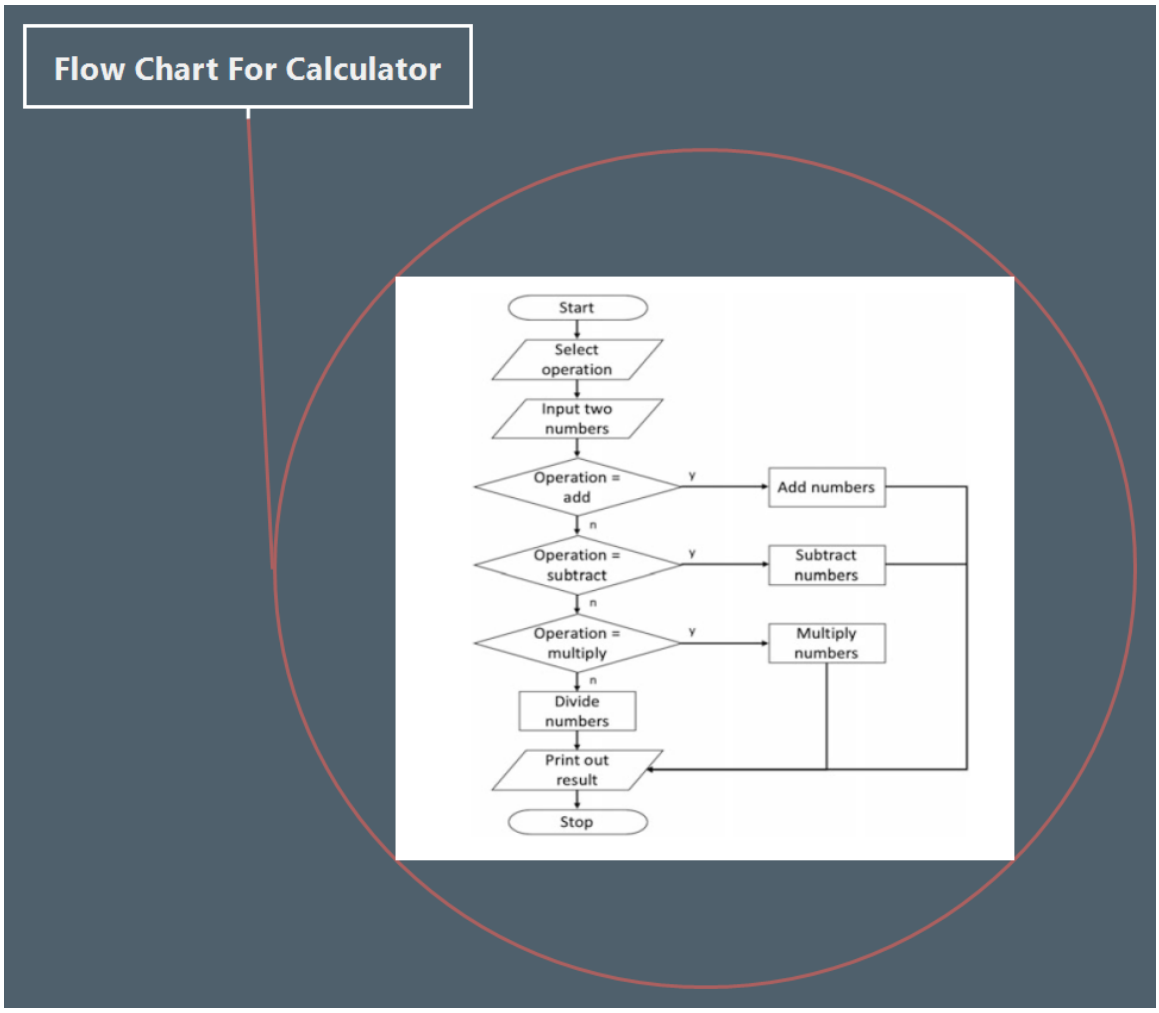


A labelled connector for use when the target is on another page (another flowchart).

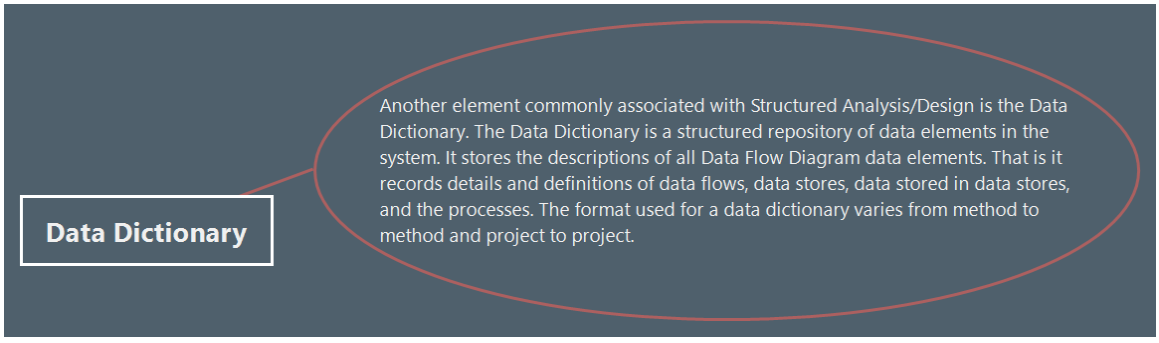
Example



Flow Chart For Calculator



5.5. Data Dictionary



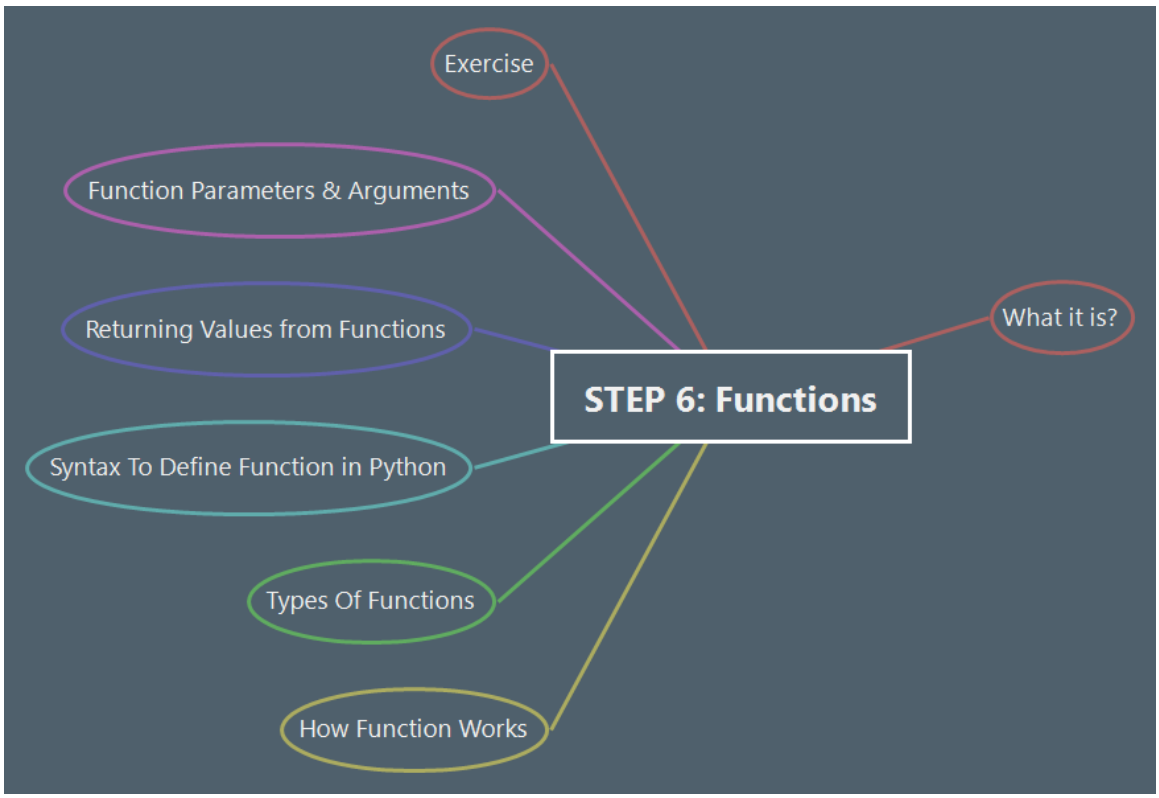
5.5.1. Another element commonly associated with Structured Analysis/Design is the Data

Dictionary. The Data Dictionary is a structured repository of data elements in the system. It stores the descriptions of all Data Flow Diagram data elements. That is it

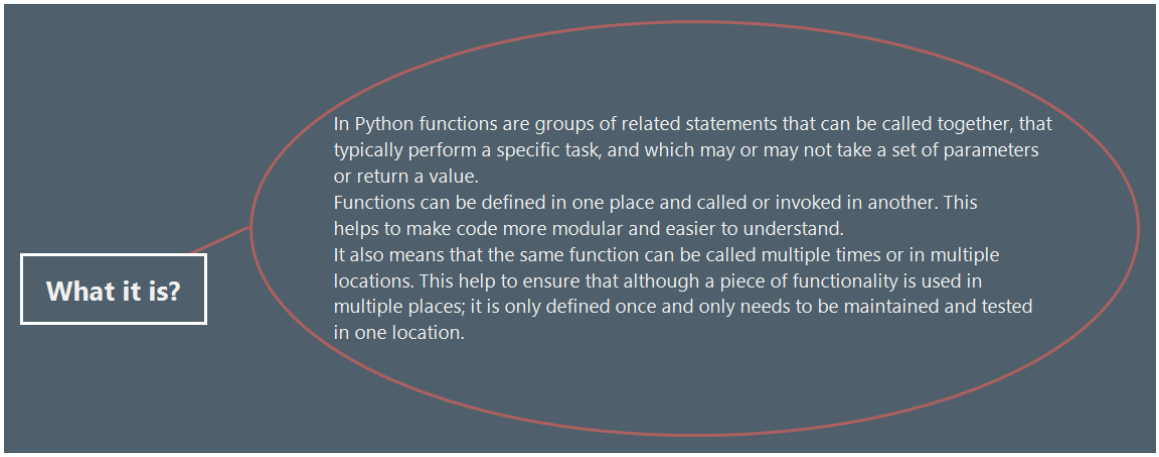
records details and definitions of data flows, data stores, data stored in data stores,

and the processes. The format used for a data dictionary varies from method to method and project to project.

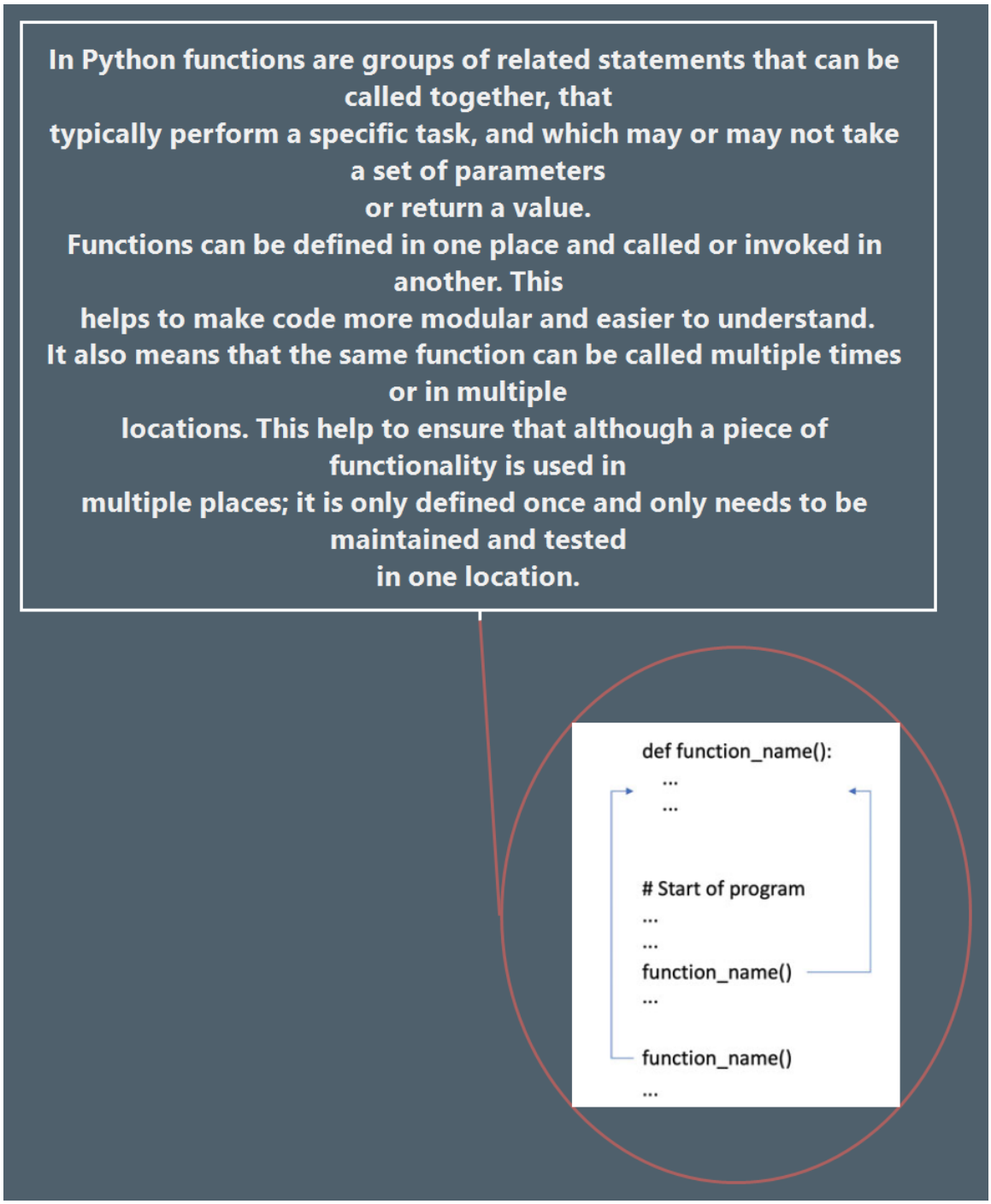
6. STEP 6: Functions



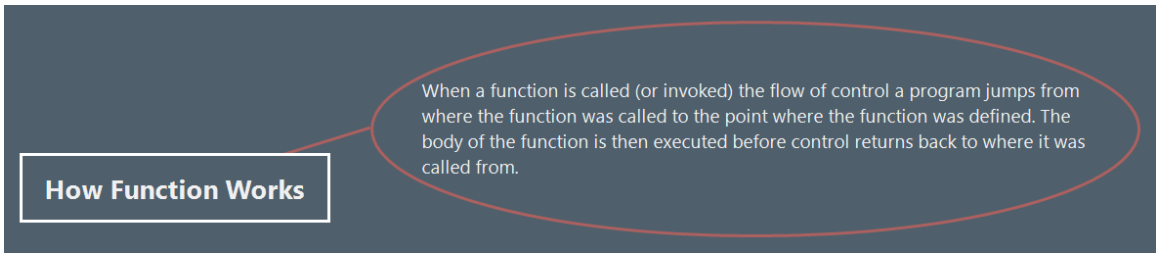
6.1. What it is?



6.1.1. In Python functions are groups of related statements that can be called together, that
typically perform a specific task, and which may or may not take a set of parameters
or return a value.
Functions can be defined in one place and called or invoked in another. This helps to make code more modular and easier to understand.
It also means that the same function can be called multiple times or in multiple locations. This help to ensure that although a piece of functionality is used in multiple places; it is only defined once and only needs to be maintained and tested
in one location.



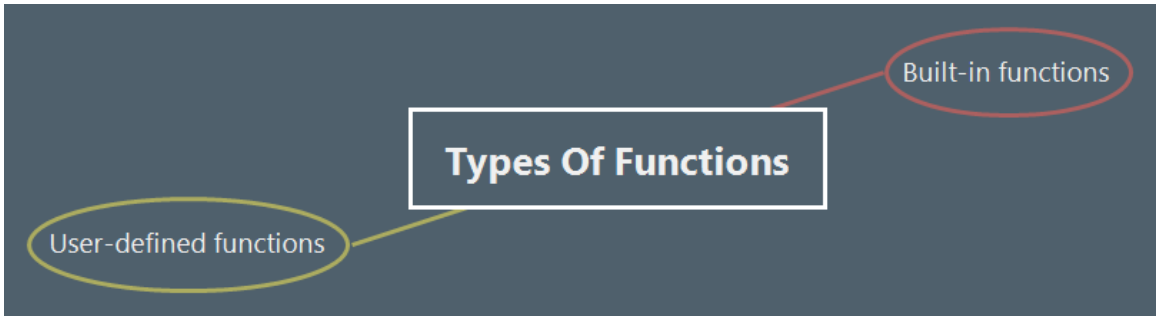
6.2. How Function Works



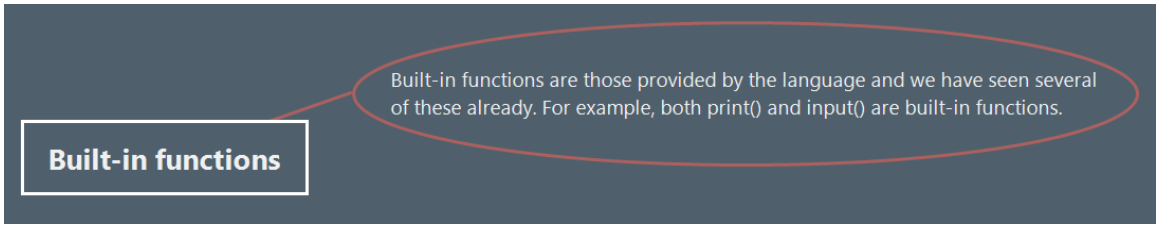
6.2.1. When a function is called (or invoked) the flow of control a program jumps from

where the function was called to the point where the function was defined. The body of the function is then executed before control returns back to where it was called from.

6.3. Types Of Functions

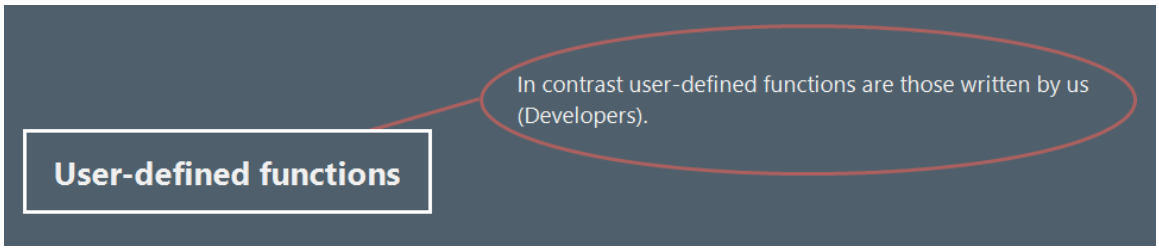


6.3.1. Built-in functions



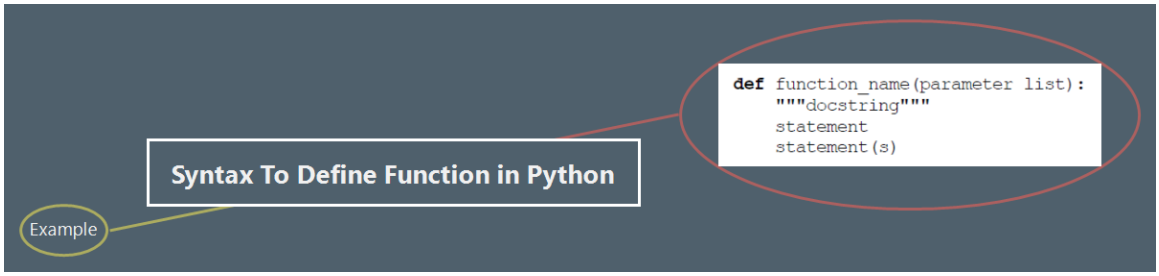
Built-in functions are those provided by the language and we have seen several of these already. For example, both print() and input() are built-in functions.

6.3.2. User-defined functions



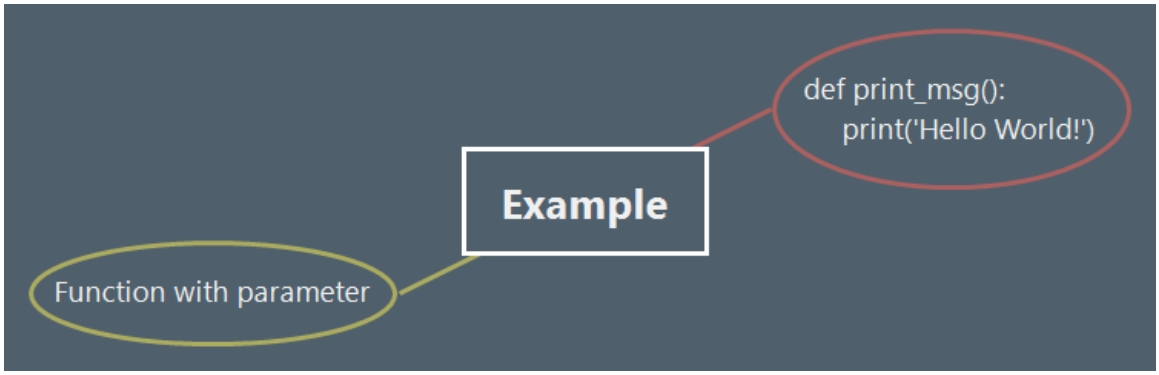
In contrast user-defined functions are those written by us (Developers).

6.4. Syntax To Define Function in Python



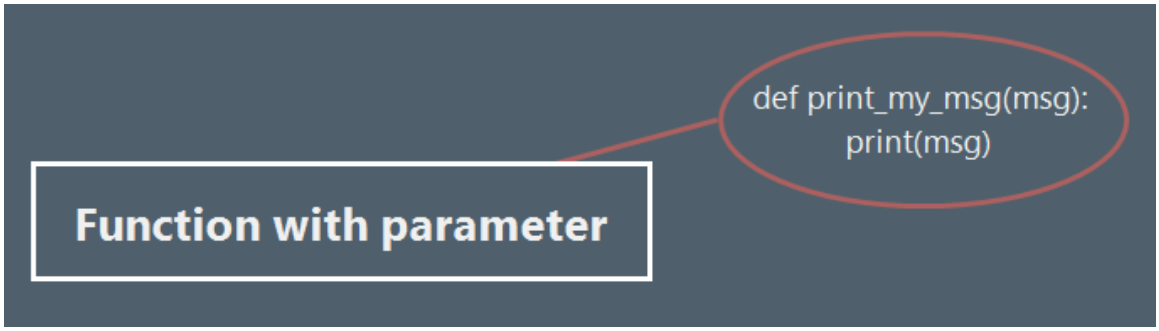
6.4.1.

6.4.2. Example



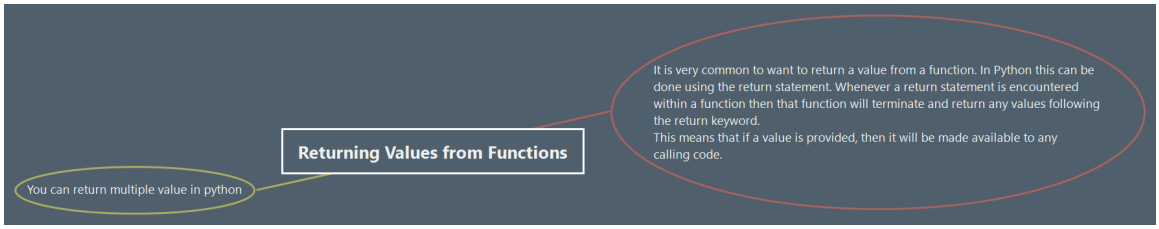

```
def print_msg():
    print('Hello World!')
```

Function with parameter

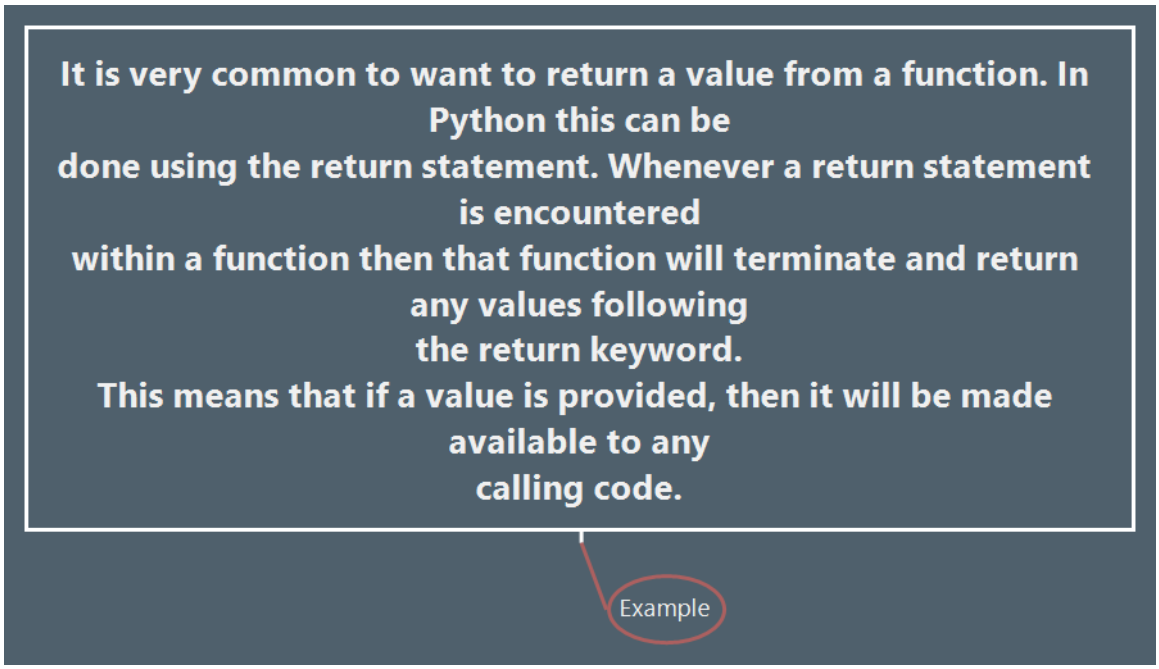



```
def print_my_msg(msg):
    print(msg)
```

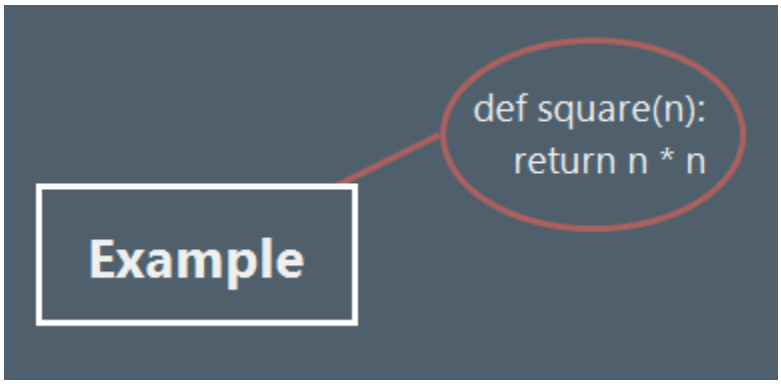
6.5. Returning Values from Functions



6.5.1. It is very common to want to return a value from a function. In Python this can be done using the return statement. Whenever a return statement is encountered within a function then that function will terminate and return any values following the return keyword.
This means that if a value is provided, then it will be made available to any calling code.

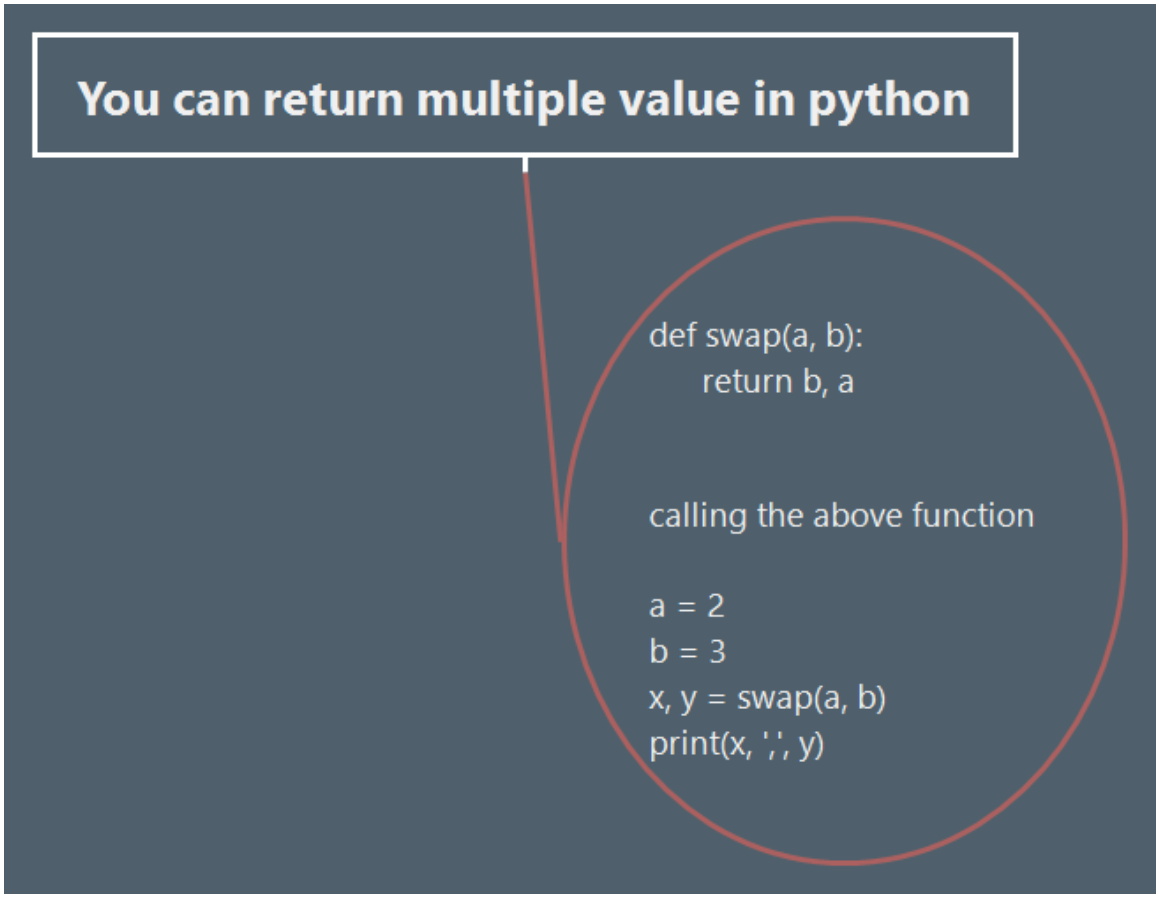


Example



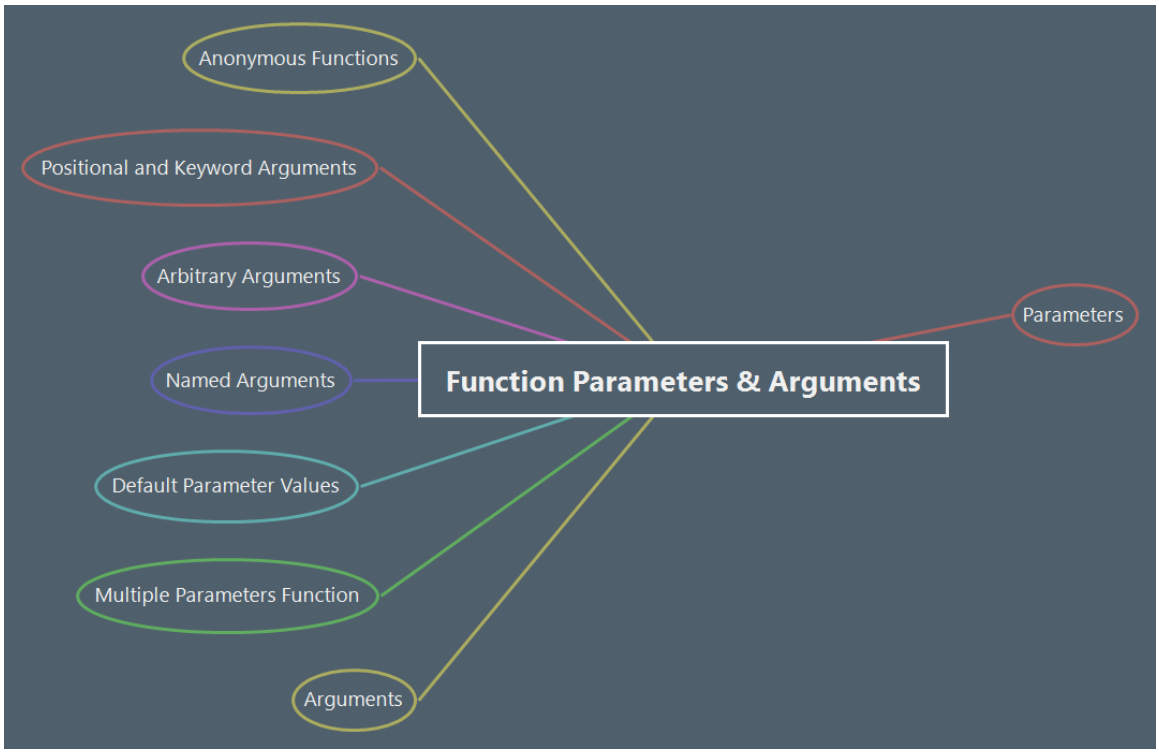

```
def square(n):
    return n * n
```

6.5.2. You can return multiple value in python

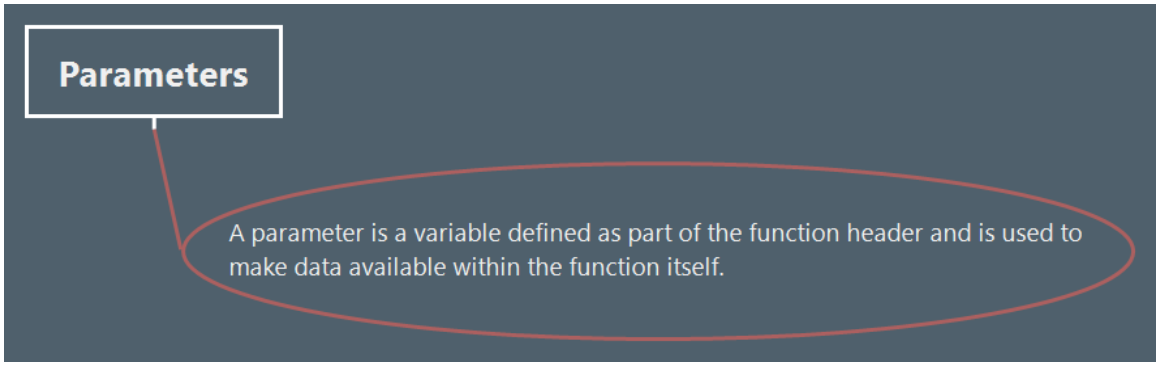



```
def swap(a, b):
    return b, a


calling the above function


a = 2
b = 3
x, y = swap(a, b)
print(x, ',', y)
```

6.6. Function Parameters & Arguments

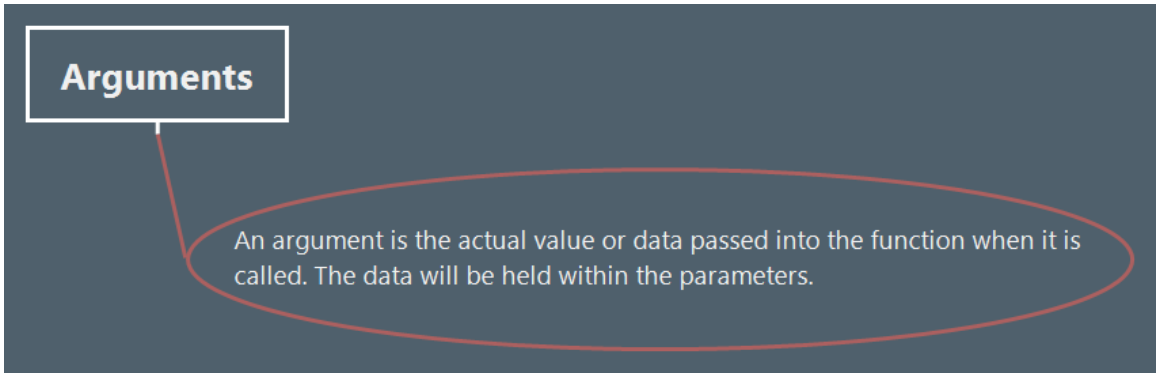


6.6.1. Parameters



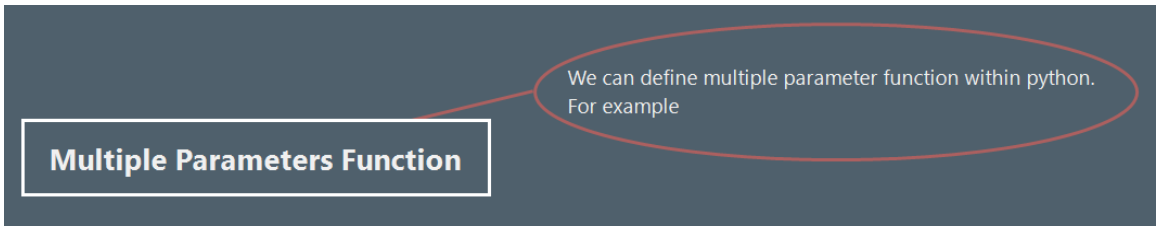
A parameter is a variable defined as part of the function header and is used to make data available within the function itself.

6.6.2. Arguments

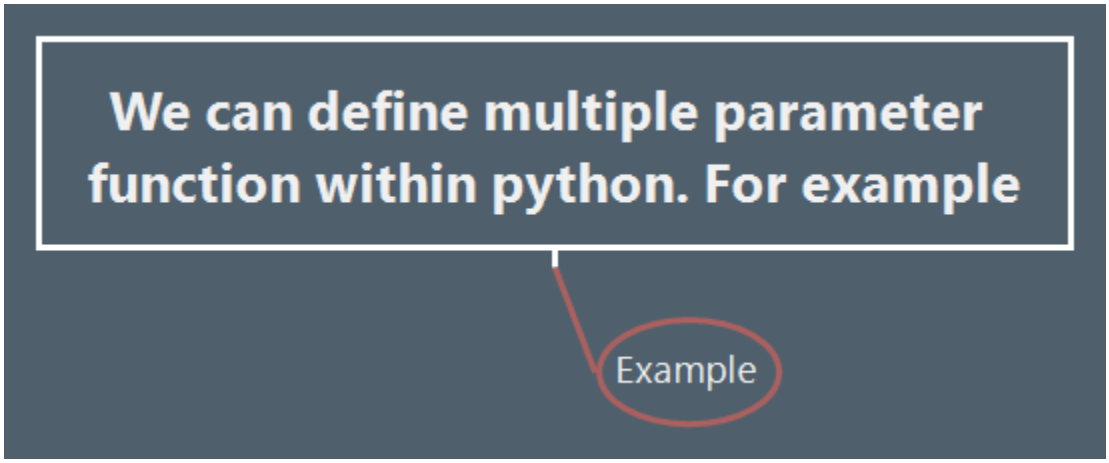


An argument is the actual value or data passed into the function when it is called. The data will be held within the parameters.

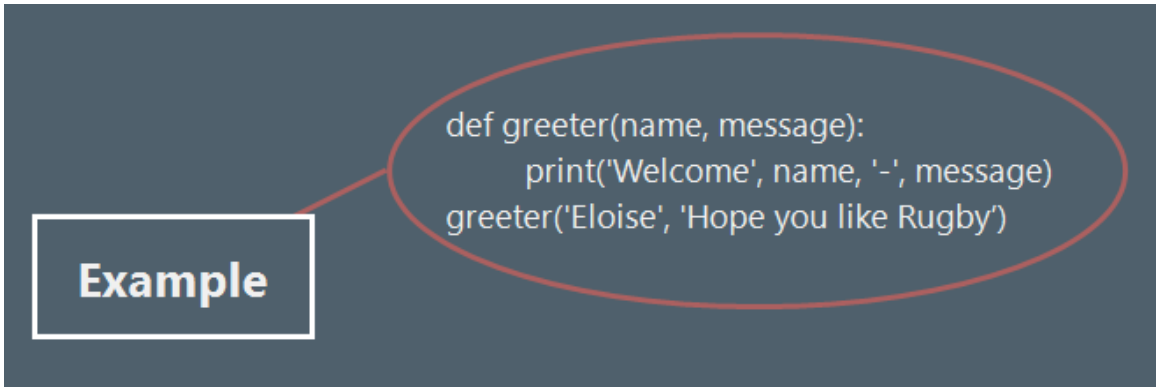
6.6.3. Multiple Parameters Function



We can define multiple parameter function within python. For example

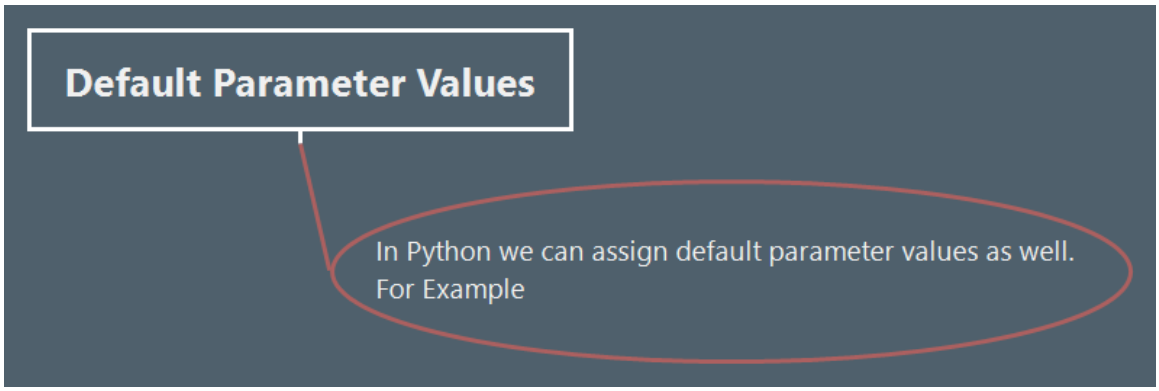


Example

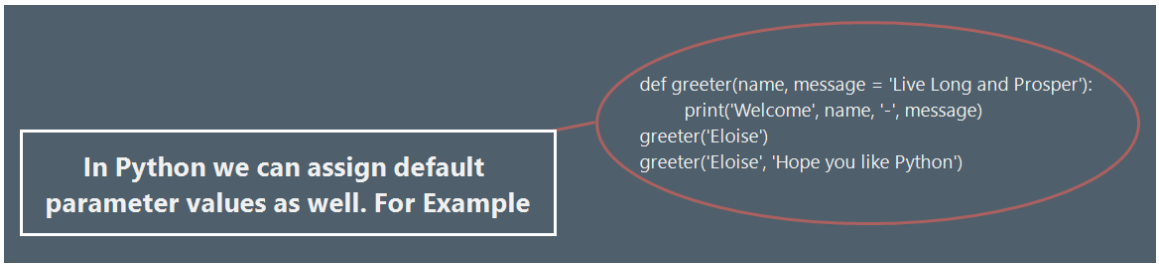



```
def greeter(name, message):
    print('Welcome', name, '-', message)
greeter('Eloise', 'Hope you like Rugby')
```

6.6.4. Default Parameter Values

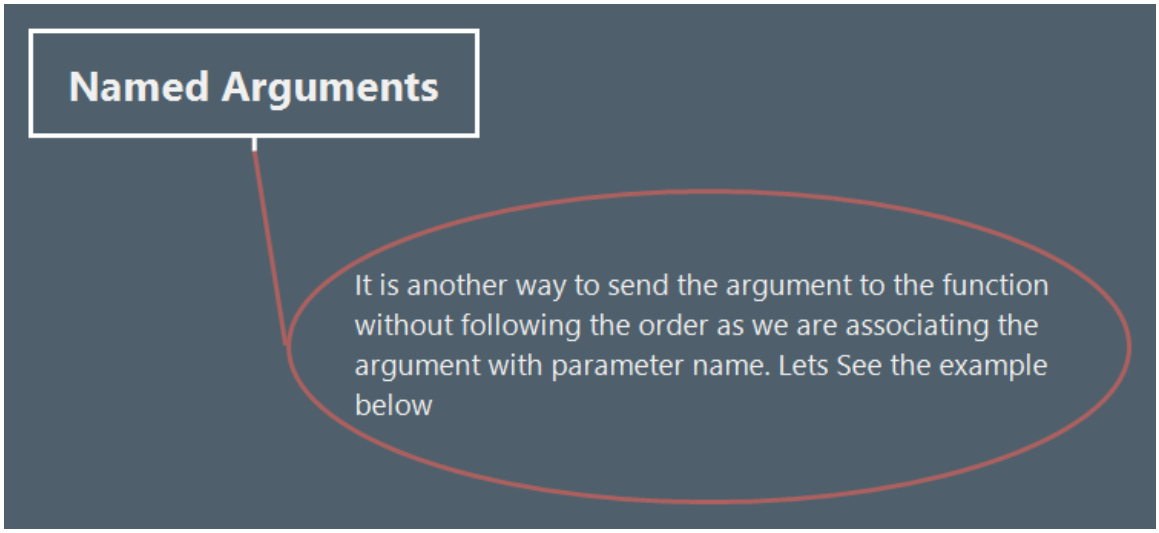


In Python we can assign default parameter values as well. For Example

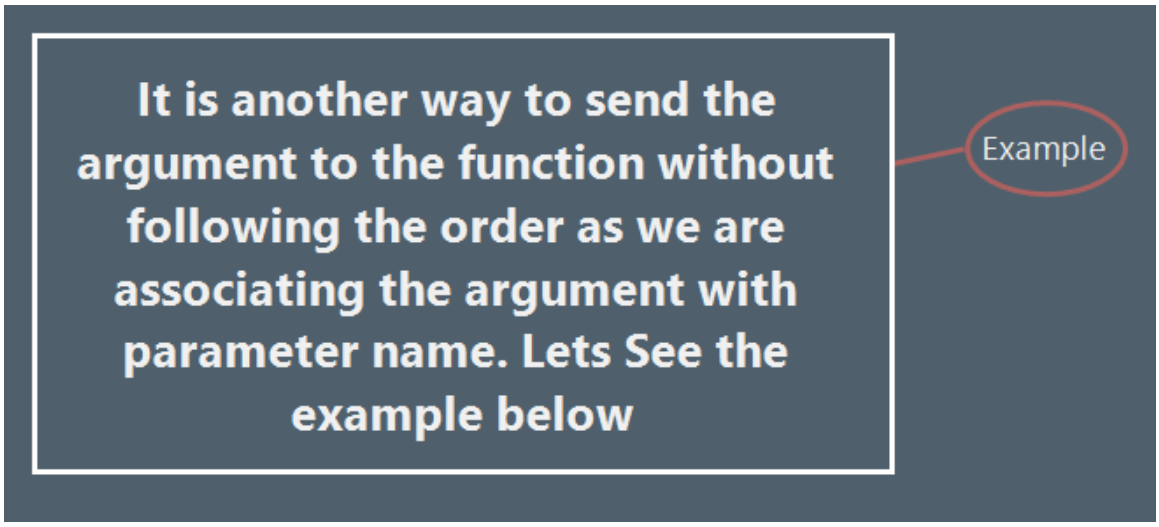



```
def greeter(name, message = 'Live Long and Prosper'):
    print('Welcome', name, '-', message)
greeter('Eloise')
greeter('Eloise', 'Hope you like Python')
```

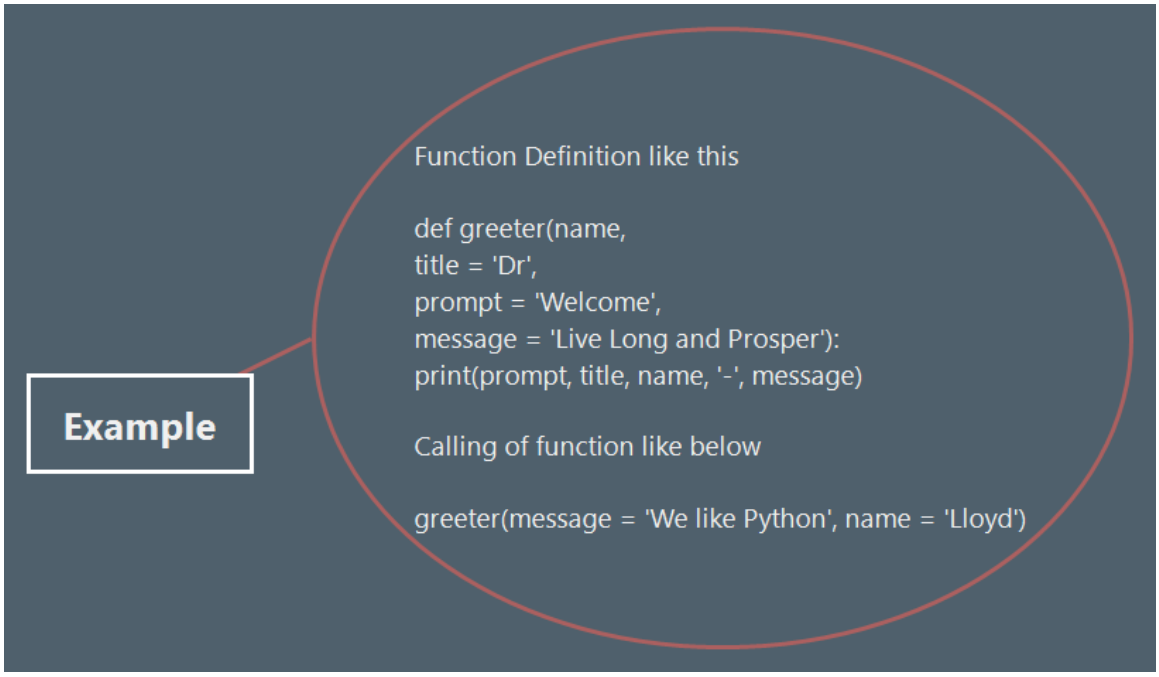
6.6.5. Named Arguments



It is another way to send the argument to the function without following the order as we are associating the argument with parameter name. Lets See the example below



Example



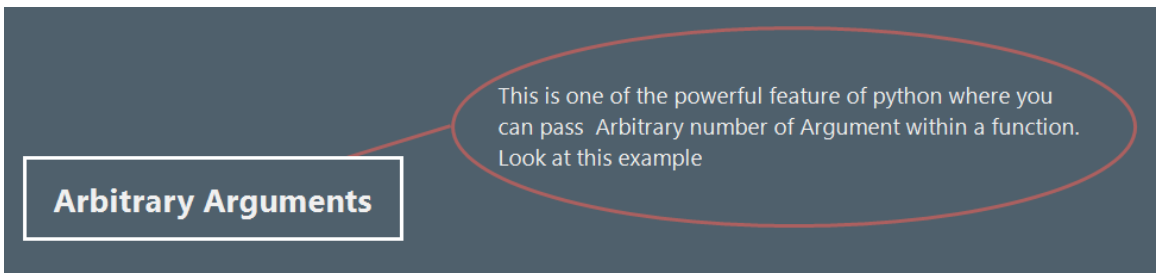
Function Definition like this

```
def greeter(name,
title = 'Dr',
prompt = 'Welcome',
message = 'Live Long and Prosper'):
print(prompt, title, name, '-', message)
```

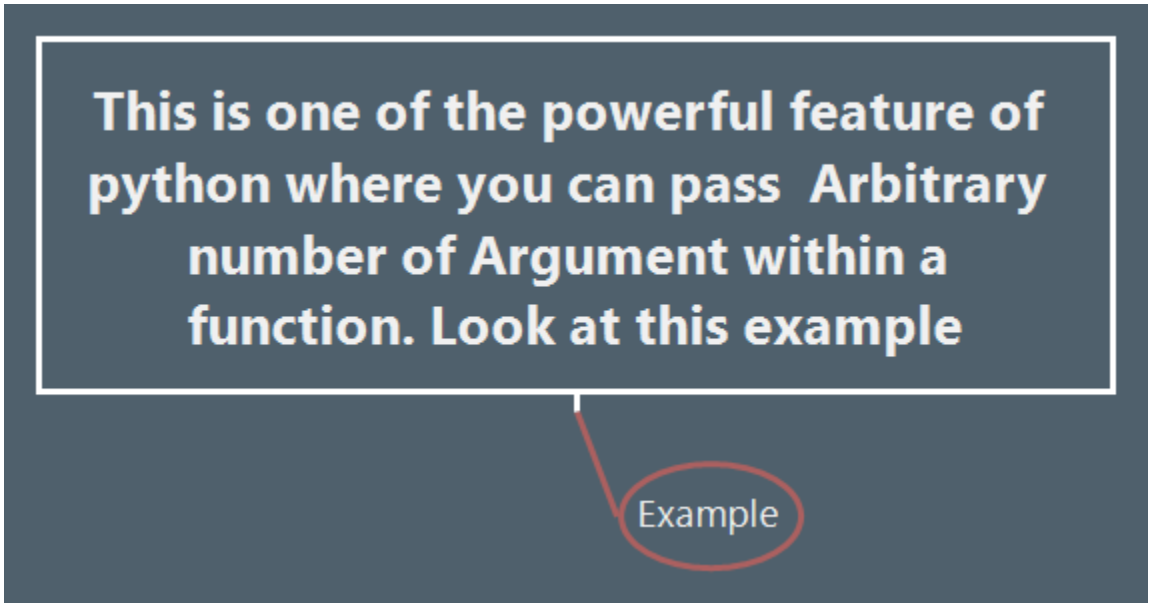
Calling of function like below

```
greeter(message = 'We like Python', name = 'Lloyd')
```

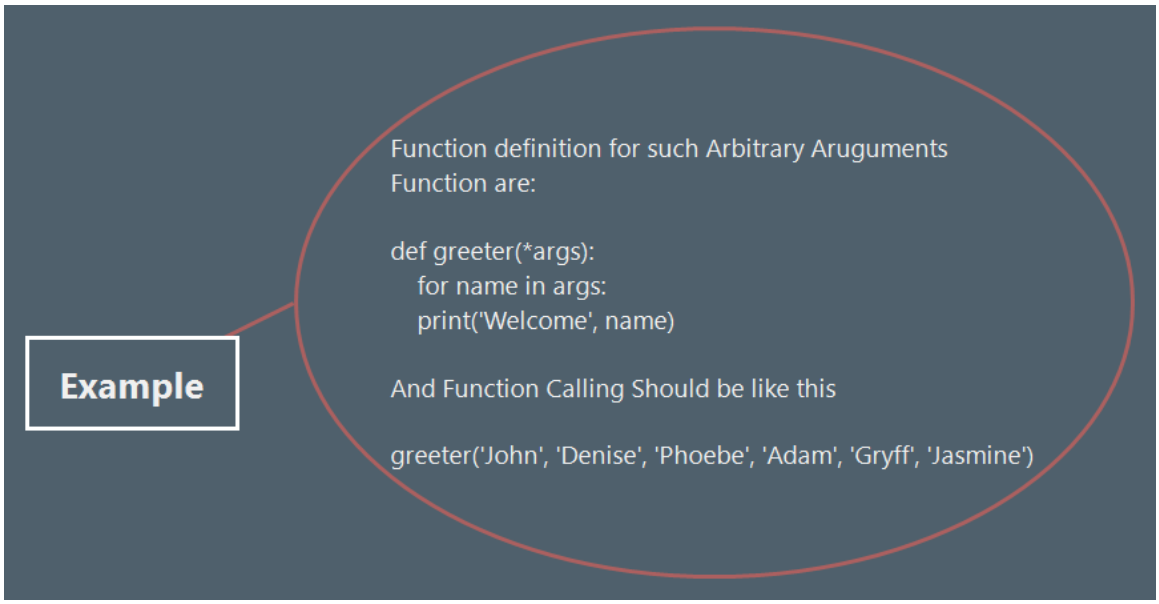
6.6.6. Arbitrary Arguments



This is one of the powerful feature of python where you can pass Arbitrary number of Argument within a function. Look at this example



Example



Function definition for such Arbitrary Aruguments Function are:

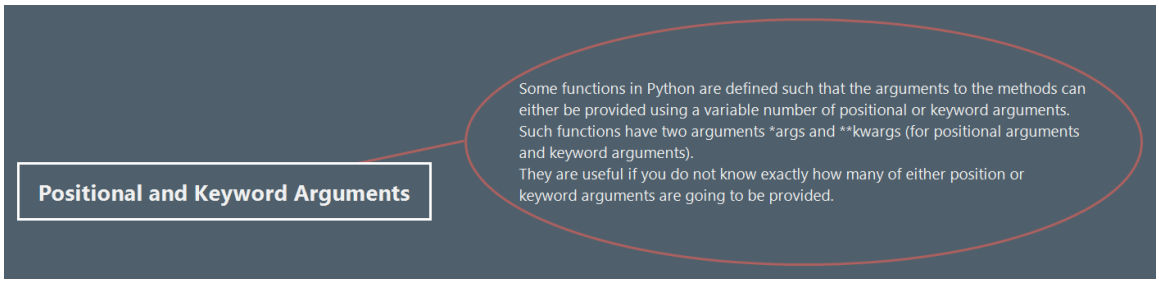
```
def greeter(*args):
   for name in args:
```

print('Welcome', name)

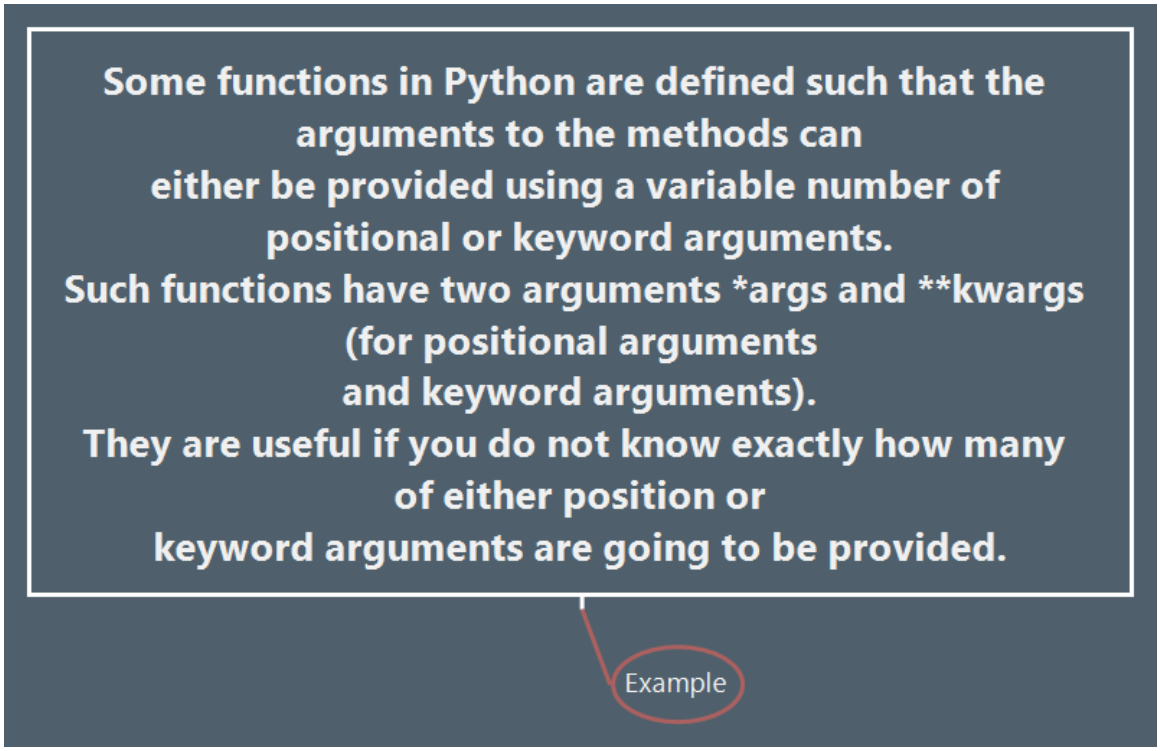
And Function Calling Should be like this

greeter('John', 'Denise', 'Phoebe', 'Adam', 'Gryff', 'Jasmine')

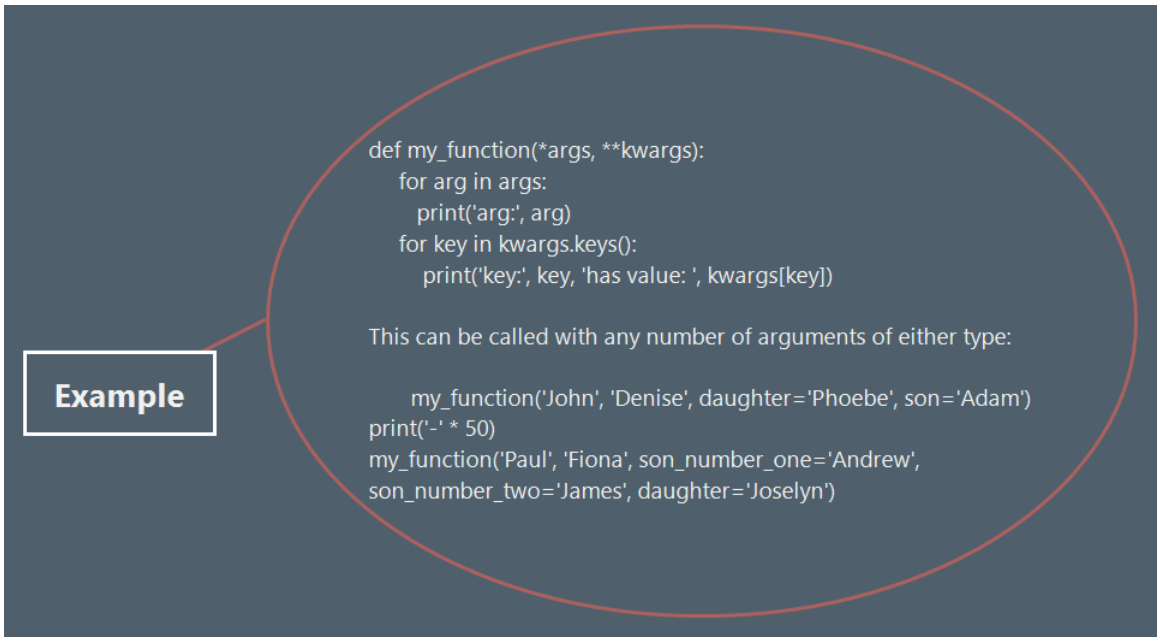
6.6.7. Positional and Keyword Arguments



Some functions in Python are defined such that the arguments to the methods can
either be provided using a variable number of positional or keyword arguments.
Such functions have two arguments *args and **kwargs (for positional arguments
and keyword arguments).
They are useful if you do not know exactly how many of either position or keyword arguments are going to be provided.



Example



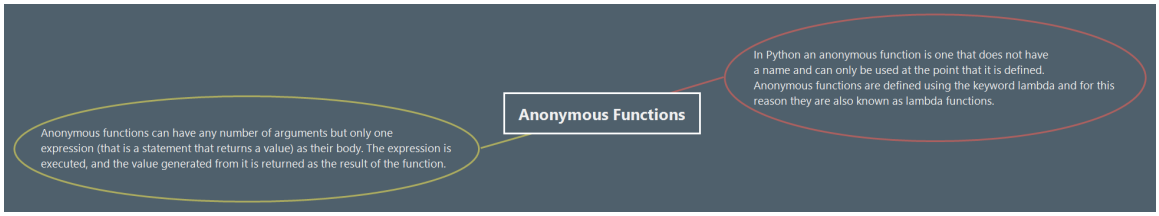

```
def my_function(*args, **kwargs):
    for arg in args:
        print('arg:', arg)
```

```
    for key in kwargs.keys():
        print('key:', key, 'has value: ', kwargs[key])
```

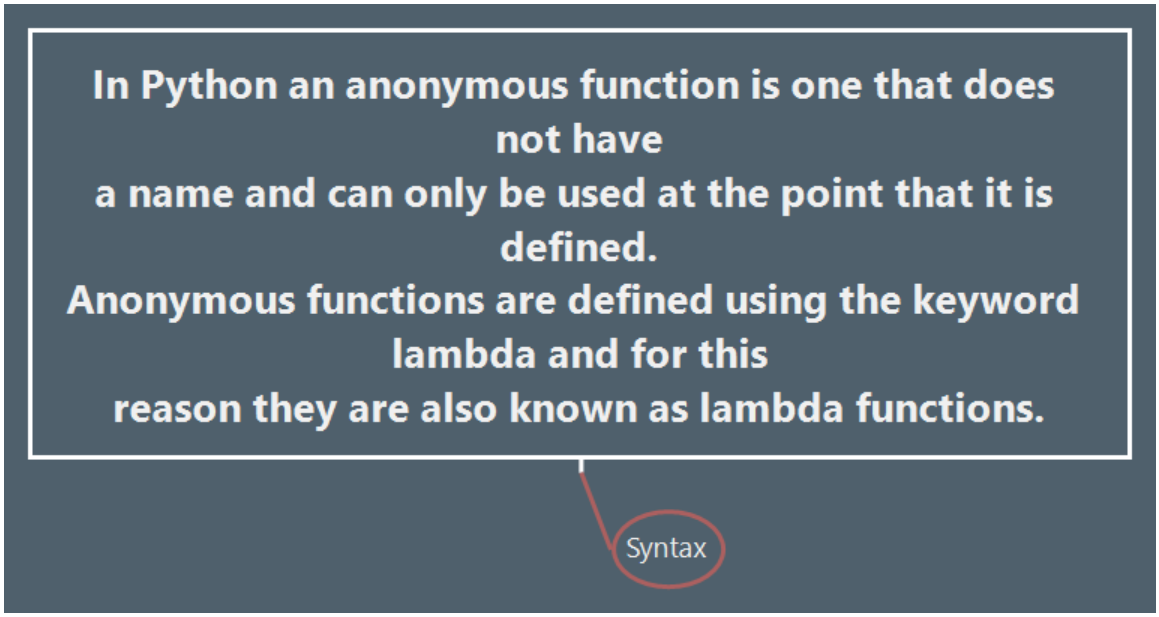
This can be called with any number of arguments of either type:

```
    my_function('John', 'Denise', daughter='Phoebe', son='Adam')
print('-' * 50)
my_function('Paul', 'Fiona', son_number_one='Andrew',
son_number_two='James', daughter='Joselyn')
```

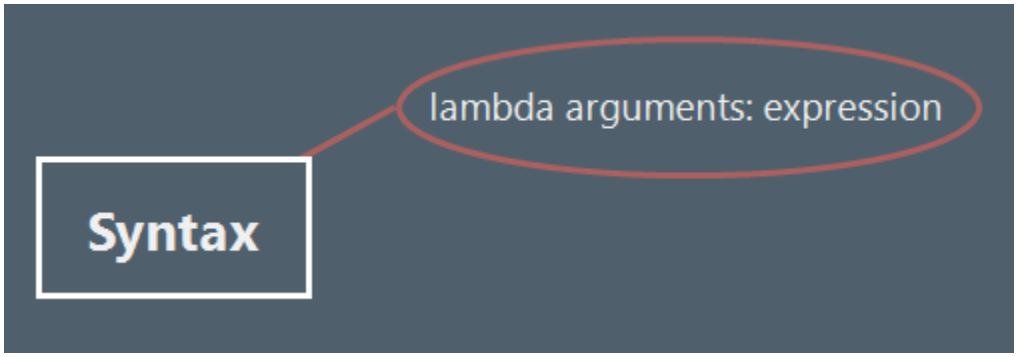
6.6.8. Anonymous Functions



In Python an anonymous function is one that does not have
a name and can only be used at the point that it is defined.
Anonymous functions are defined using the keyword lambda and for this reason they are also known as lambda functions.

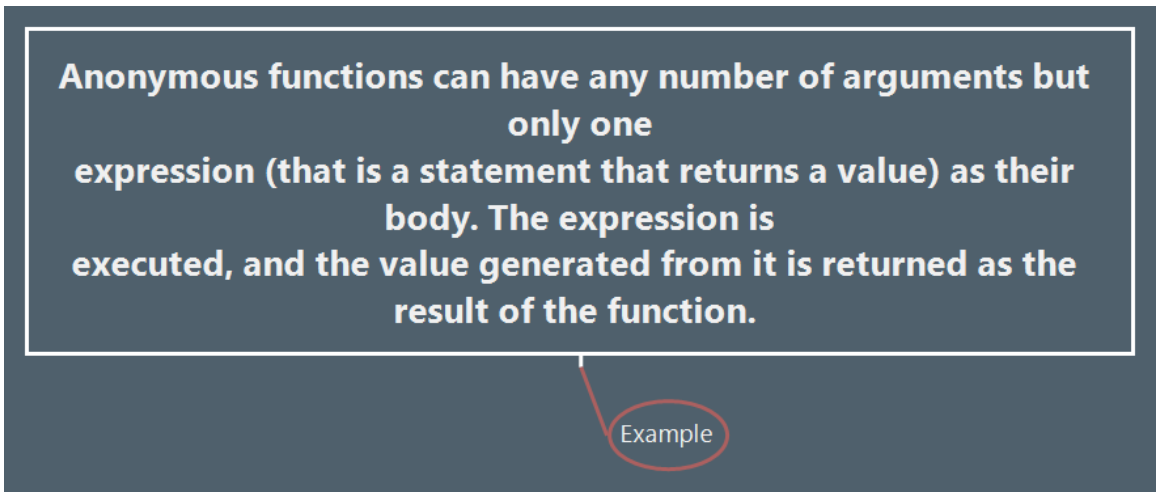


Syntax

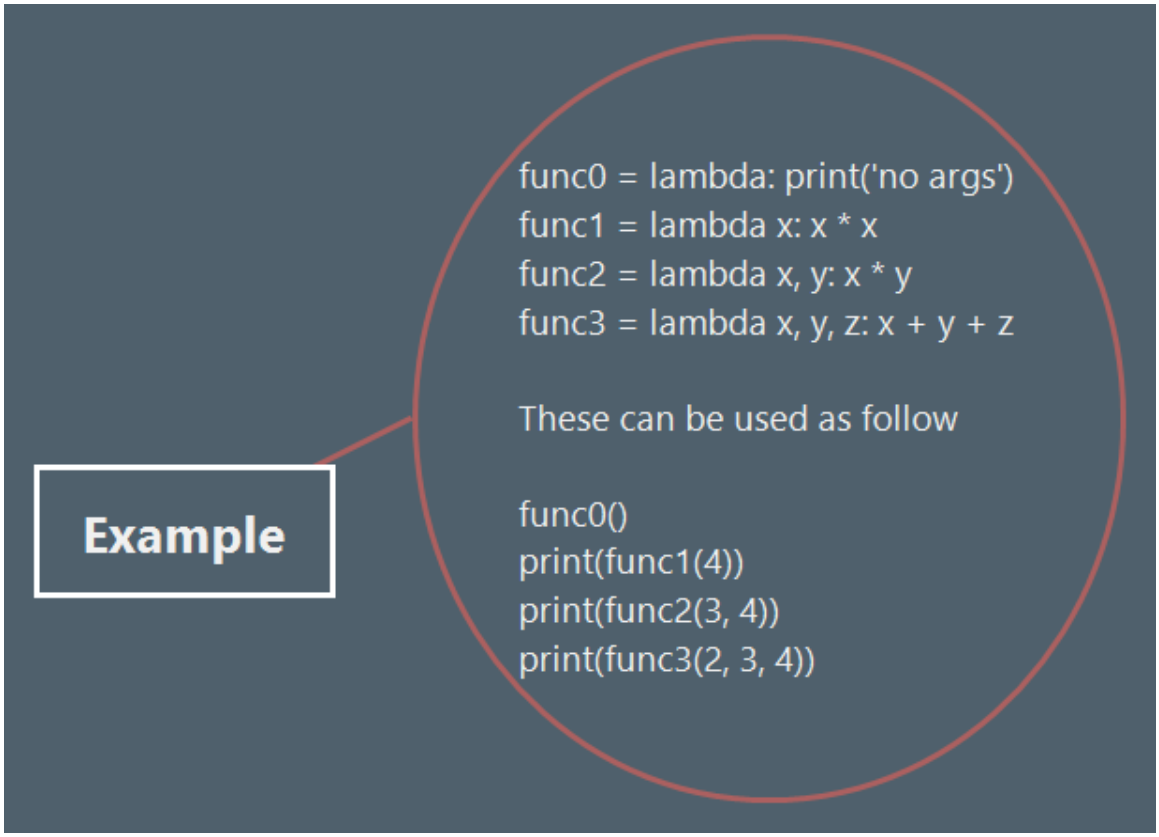


lambda arguments: expression

Anonymous functions can have any number of arguments but only one expression (that is a statement that returns a value) as their body. The expression is executed, and the value generated from it is returned as the result of the function.

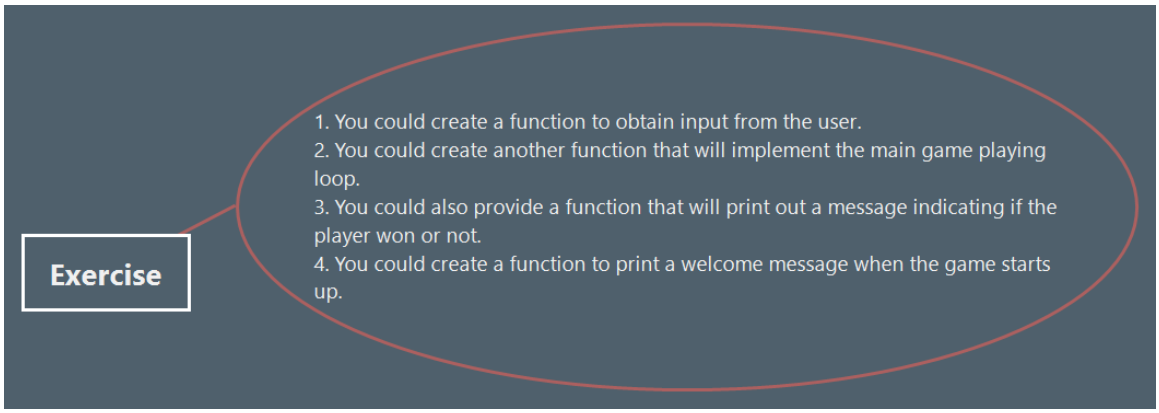


Example




```
func0 = lambda: print('no args')
func1 = lambda x: x * x
func2 = lambda x, y: x * y
func3 = lambda x, y, z: x + y + z

These can be used as follow

func0()
print(func1(4))
print(func2(3, 4))
print(func3(2, 3, 4))
```

6.7. Exercise



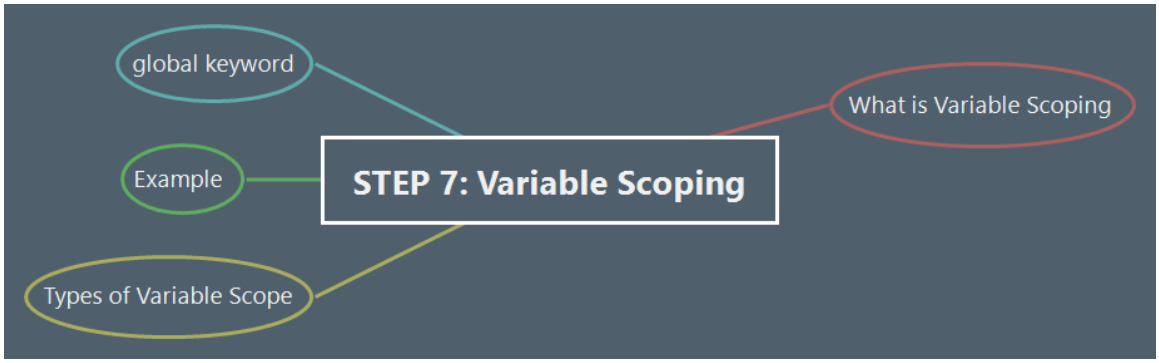
6.7.1. 1. You could create a function to obtain input from the user.

2. You could create another function that will implement the main game playing loop.

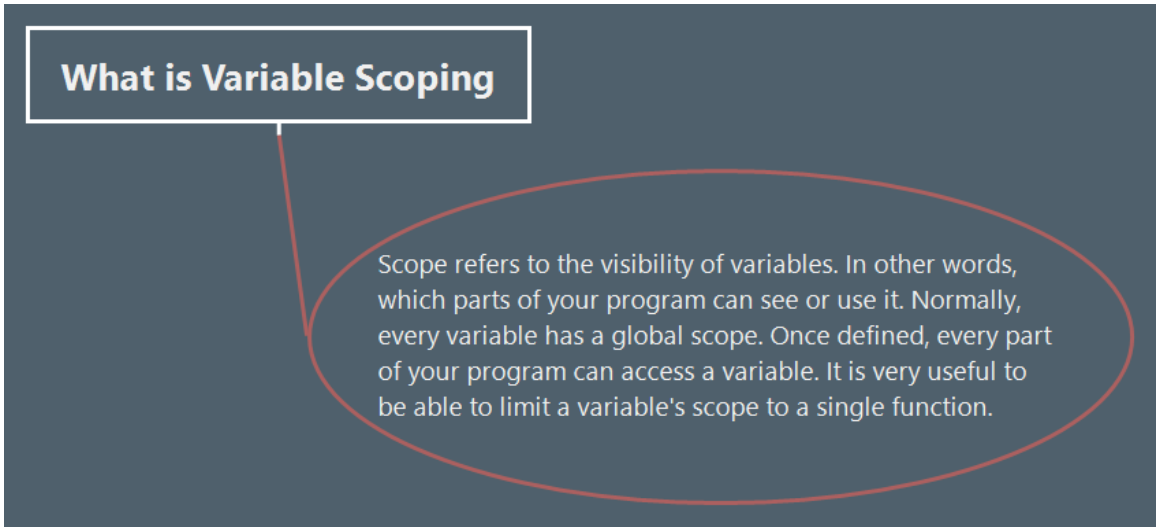
3. You could also provide a function that will print out a message indicating if the player won or not.

4. You could create a function to print a welcome message when the game starts up.

7. STEP 7: Variable Scoping

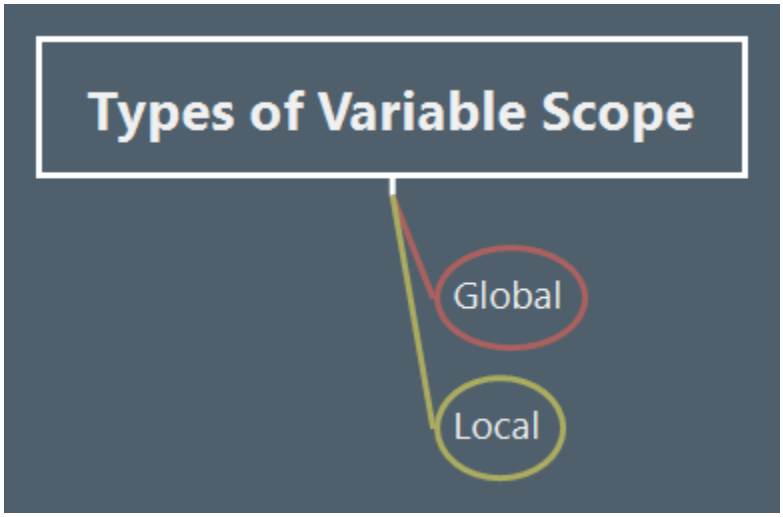


7.1. What is Variable Scoping

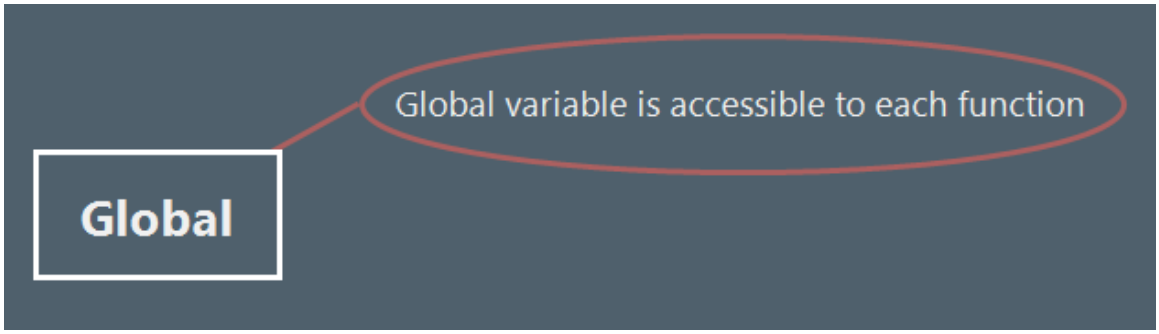


7.1.1. Scope refers to the visibility of variables. In other words, which parts of your program can see or use it. Normally, every variable has a global scope. Once defined, every part of your program can access a variable. It is very useful to be able to limit a variable's scope to a single function.

7.2. Types of Variable Scope

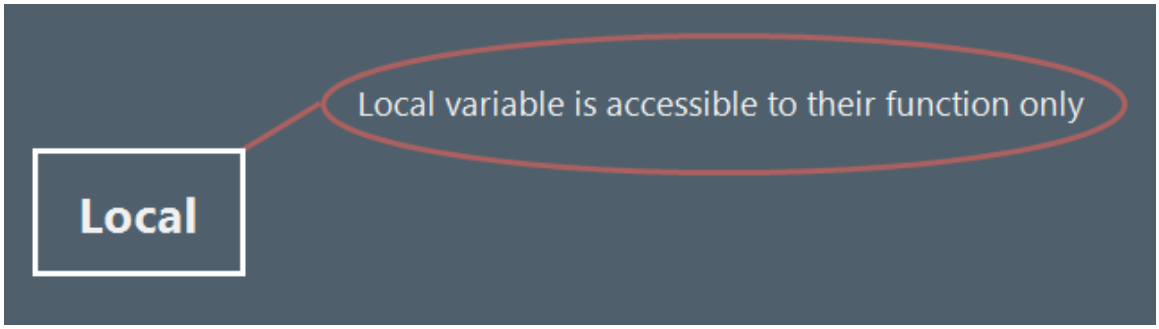


7.2.1. Global



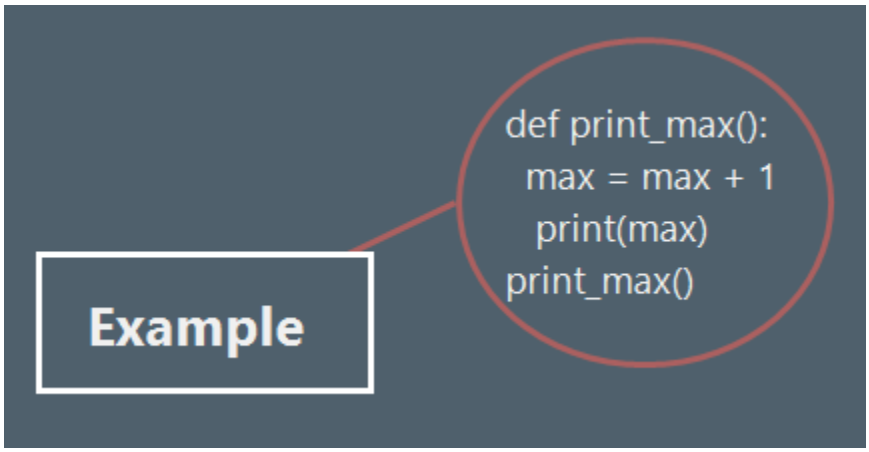
Global variable is accessible to each function

7.2.2. Local



Local variable is accessible to their function only

7.3. Example

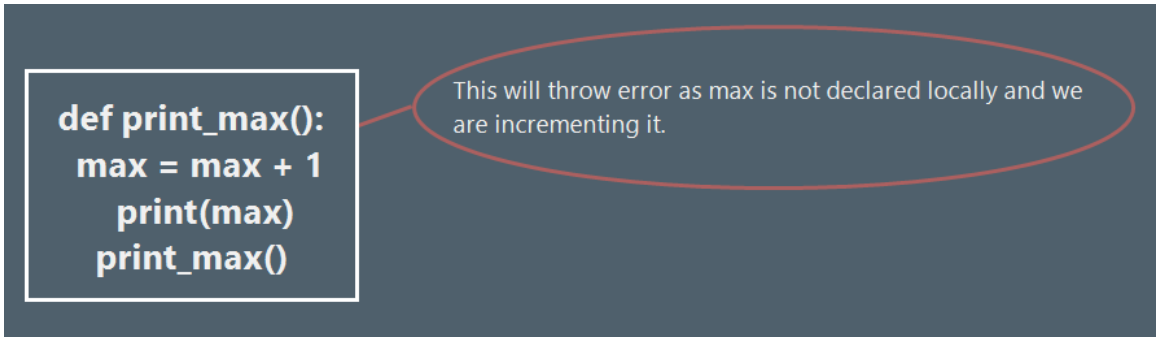


7.3.1. def print_max():

max = max + 1

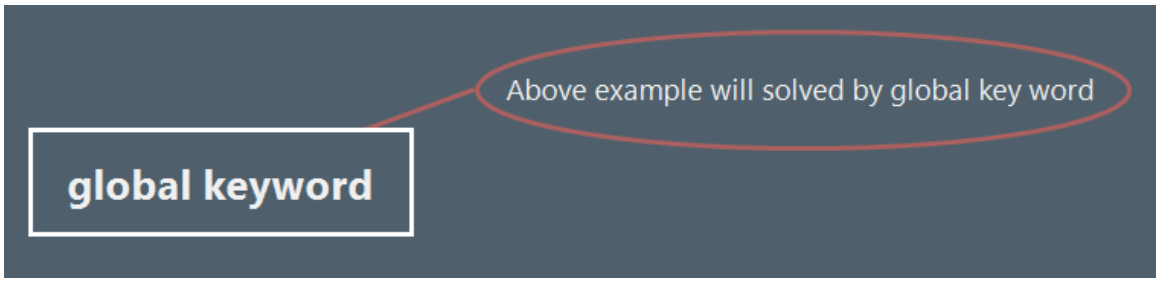
print(max)

print_max()

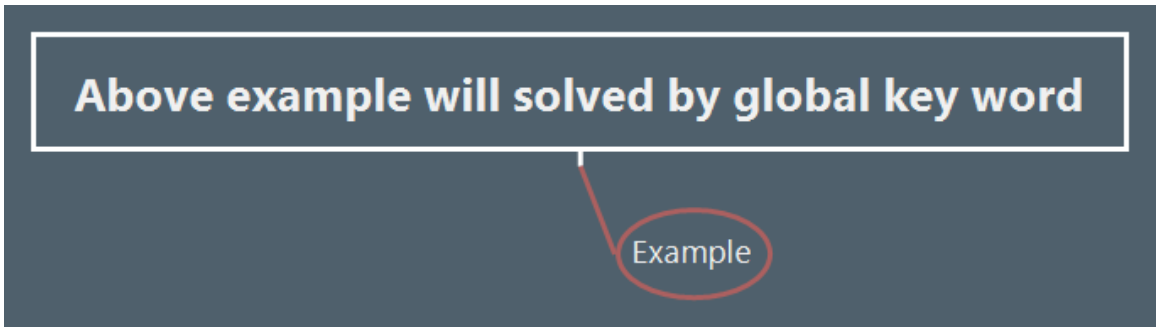


This will throw error as max is not declared locally and we are incrementing it.

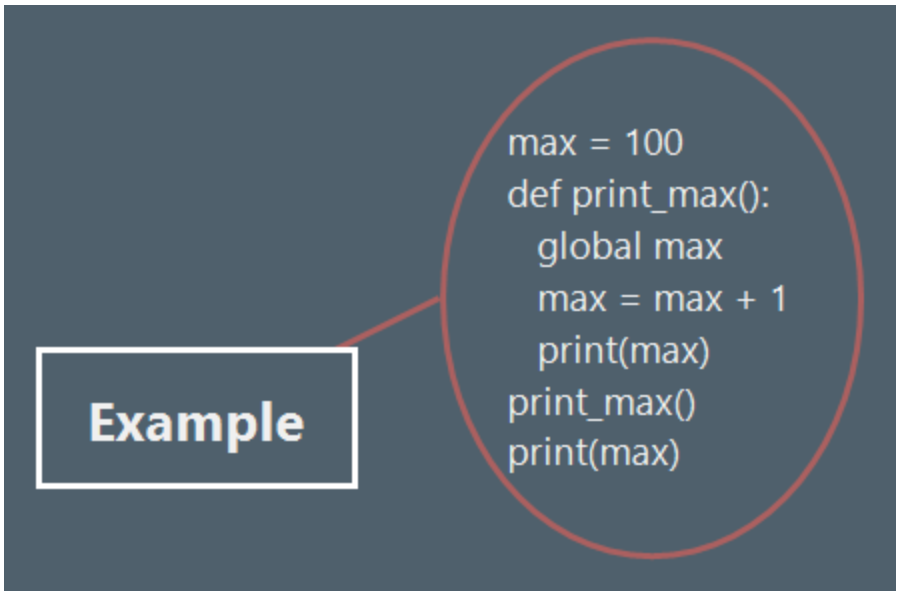
7.4. global keyword



7.4.1. Above example will solved by global key word

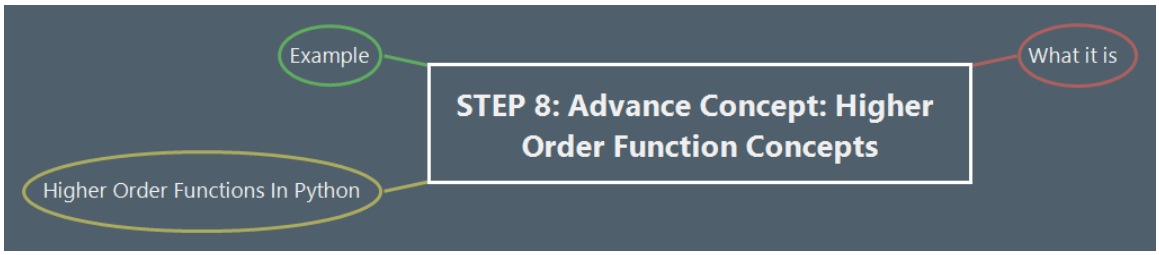


Example

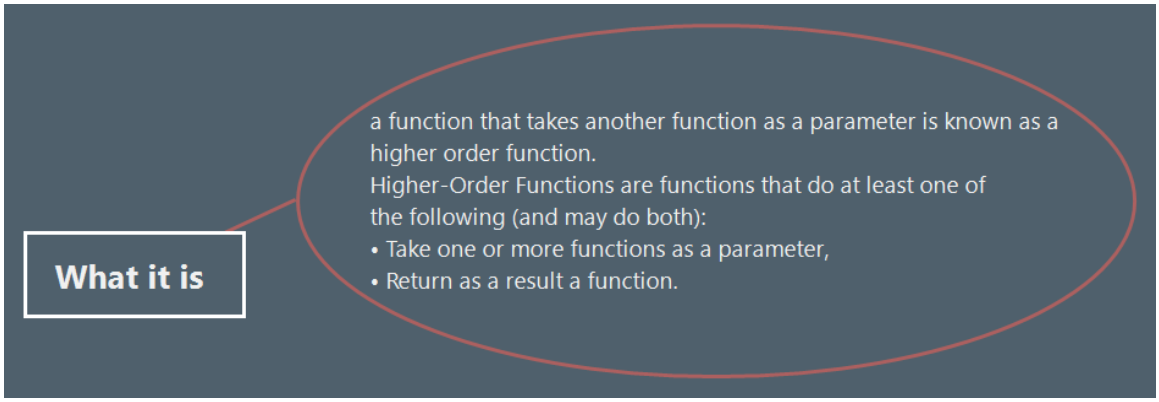



```
max = 100
def print_max():
  global max
  max = max + 1
  print(max)
print_max()
print(max)
```

8. STEP 8: Advance Concept: Higher Order Function Concepts



8.1. What it is

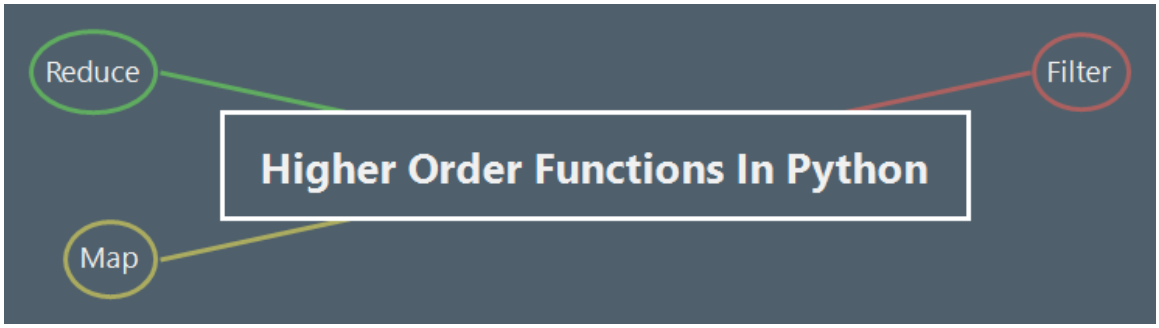


8.1.1. a function that takes another function as a parameter is known as a higher order function.

Higher-Order Functions are functions that do at least one of the following (and may do both):

- **Take one or more functions as a parameter,**
- **Return as a result a function.**

8.2. Higher Order Functions In Python

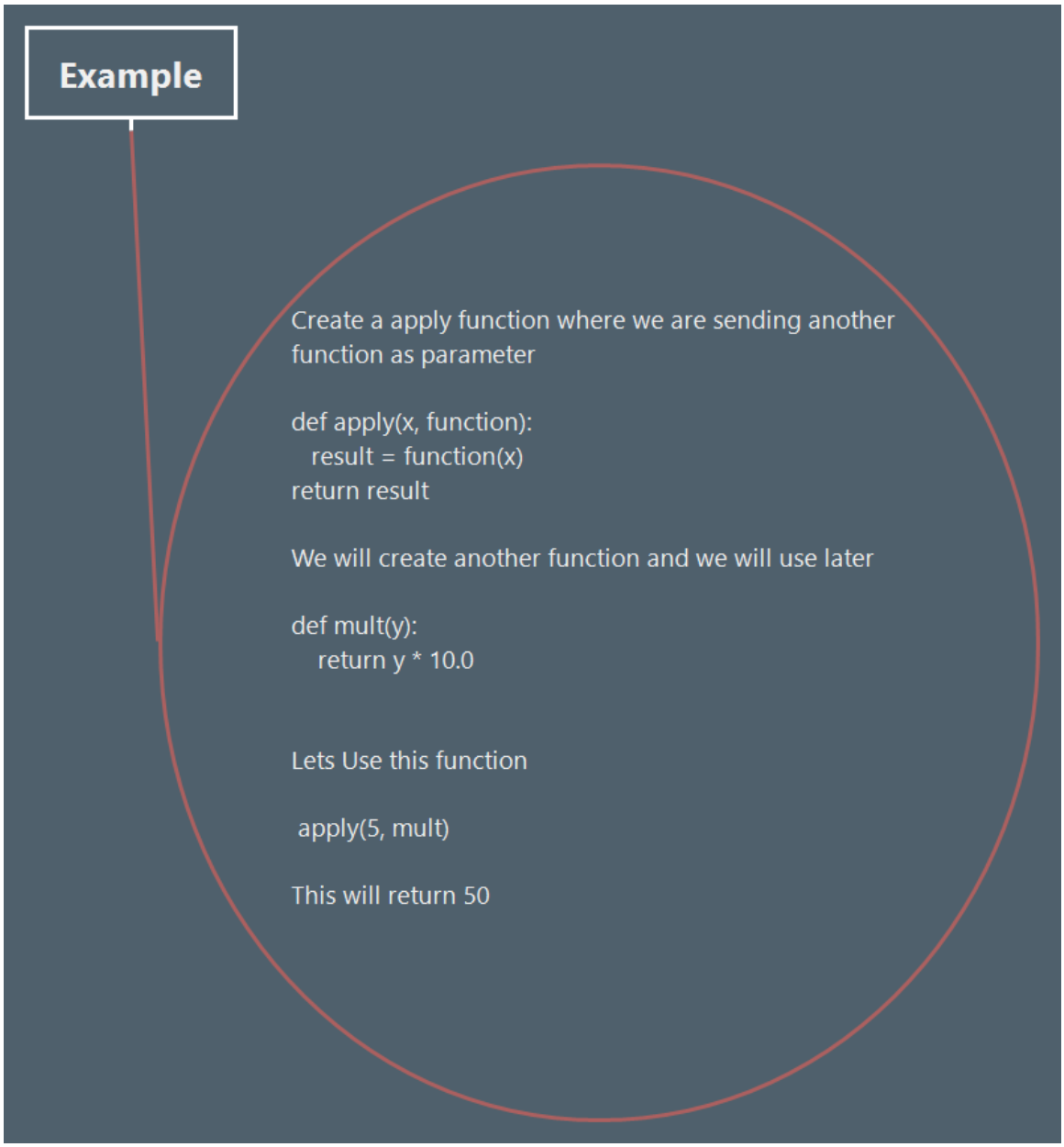


8.2.1. Filter

8.2.2. Map

8.2.3. Reduce

8.3. Example



8.3.1. Create a apply function where we are sending another function as parameter

```
def apply(x, function):
  result = function(x)
return result
```

We will create another function and we will use later

```
def mult(y):
    return y * 10.0
```

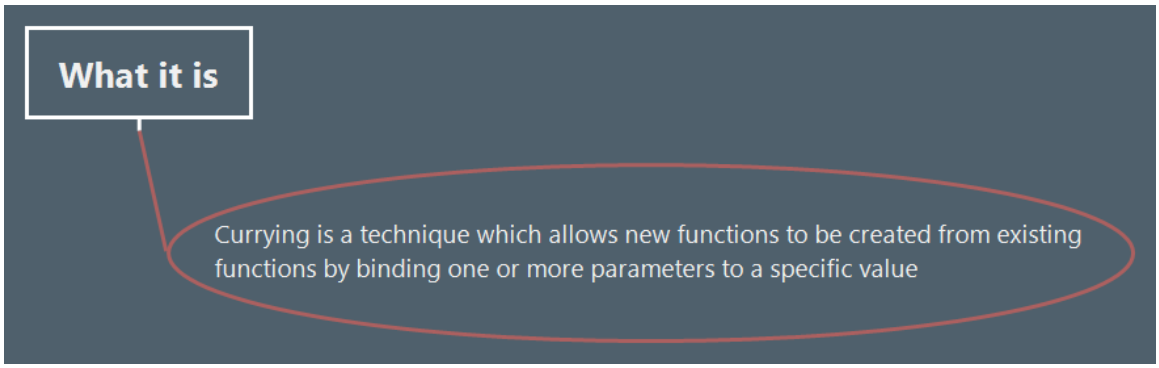
Lets Use this function

```
apply(5, mult)
```

This will return 50

9. STEP 9: Advance Concept: Curried Function

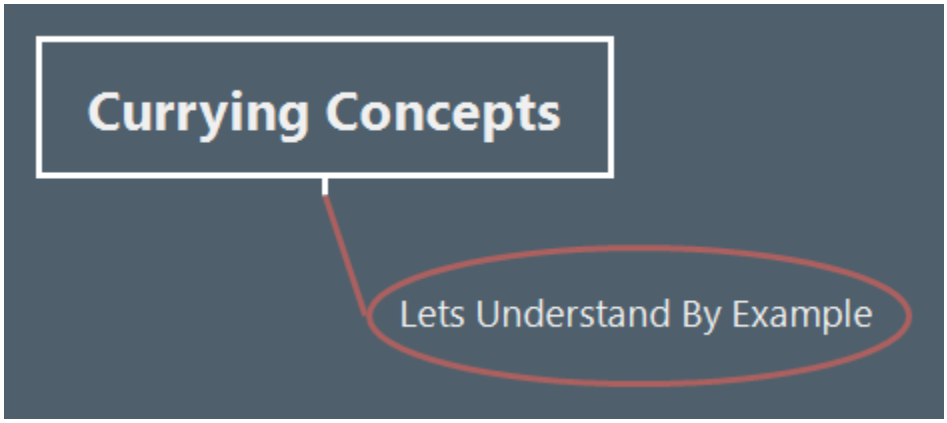
9.1. What it is



9.1.1. Currying is a technique which allows new functions to be created from existing

functions by binding one or more parameters to a specific value

9.2. Currying Concepts



9.2.1. Lets Understand By Example

Suppose if we have a function

operation(x, y): return x * y

the function can be used as

operation(2, 5)
operation(2, 10)
operation(2, 6)
operation(2, 151)

all of the above double the number

but we have to provide 2 for every call and that is not changing this would mean that we can create a function which takes 1 parameters. So to that we can use currying techniques in python as follow

double = operation(2 , *)

now we call this function like

double(2)
double(151)
double(200)