

Study Unit 3

Flow control, functions and libraries in Python Outline

- The if and else statement
- for loop and while loop
- continue and break statements
- Python functions
- Python modules and packages

Study Unit Duration

This Study Unit requires a minimum of 3 hours' formal study time.

You may spend an additional 2-3 hours on revision.

Flow Control, Loops and Functions in Python



Preamble

In the eyes of most philosophers of programming, the one thing that separates programming from the early days' automation - such as the Jacquard loom - is its ability to make decisions as to what to do next. In the programming world, that is known as flow control - control of the "flow" of executed code.

In this study unit, you will learn conditions (branching) statements in python. Conditions (branching) are known as control structures and they determine whether a block of code will run or not. You will also learn Python loops, functions and how to import different packages in Python.

Learning Outcomes of Study Unit 3

Upon completion of this study unit, you should be able to:

- 3.1 Employ control statement to make decisions in Python
- 3.2 Create functions to solve problems with Python
- 3.3 Import different packages in Python to do some tasks



Terminologies, Acronyms and their Meaning

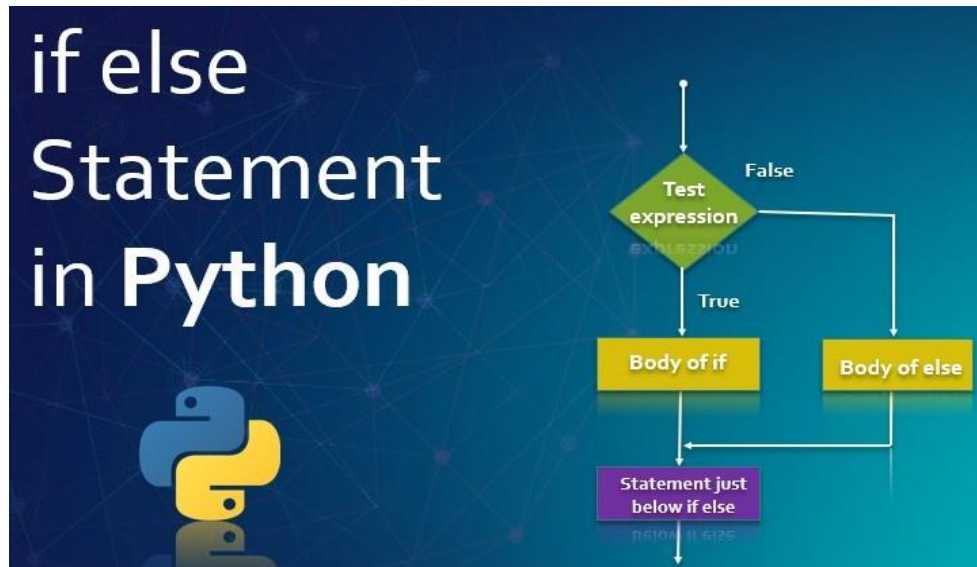
.ipynb	Jupyter notebook or Jupyter lab extension
.py	Python extension
IDE	Integrated Development environment (IDE)
R	R programming language
EDA	Exploratory Data Analysis
NaN	Not a Number

np	NumPy
pd	Pandas
os	Operating system
AI	Artificial Intelligence
int	Integer data type
str	String data type
bool	Boolean data type

3.1 Conditional Statements in Python

Majority of the time when programming, we would need to control the flow of our logic. Our program will want to perform an action in only certain cases, we can use the **if**, **elif**, and **else** statements to control for these cases. Let's work through some examples:

3.1.1 The if and else statement



The format for an if statement is

if expression:

```
Statement 1
```

else:

```
Statement 2
```

The if statement checks whether the expression evaluates to **True**, and if so, the statement 1 is executed, otherwise the statement 2 is executed. Note the **indentation** before the statement 1 and 2 and the colon (:) in the **if expression** and **else**.



All the comparison operators discussed in [unit 2](#) will be useful here.

Example 1

```
x = 10
y = 22
if (x < y):
    print("x is less than y")
else:
    print("x is greater than y")
```

x is less than y

Example 2

We can also use the print formatting option:

```
x = 10
y = 22
if (x < y):
    print(f"x = {x} is less than y = {y}")
else:
    print("x = {x} is greater than y = {y}")
```

x = 10 is less than y = 22

Example 3

Write a python **if else** statement that checks whether a given variable is an even or an odd number.

```
x = 7
if (x % 2 == 0):
    print("x is an even number")
else:
    print("x is an odd number")
```

x is an odd number

Example 4

Write a python **if else** statement that checks whether a given value is even or odd number.

Solution

We are going to use **input()** function to ask for a value, and as you know, the data type of a variable gotten from an input function is always a string. Therefore, we are going to convert the result to an integer by using **int()** function

```
x = int(input("Enter any number you want to check for odd or even"))

if (x % 2 == 0):
    print("The number you entered is an even number")
else:
    print("The number you entered is an odd number")
```

Enter any number you want to check for odd or even

The number you entered is an odd number

Example 5: Game of guess

You want to know whether your friends can guess your ATM password accurately. You wrote the below code and then ask each and every one of them to enter your password.

```
saved_password = 1519

new_password = input("Please guess my password")

if int(new_password) == saved_password:
    print("Password correct")
else:
    print("The password you enter is not correct")
```

Please guess my password

Example 6

The password you enter is not correct

Test if a is less than

b, AND if c is greater than a:

```
a = 10
b = 13
c = 16
if (a > b and c > a):
    print("Both conditions are True")
else:
    print("None of the conditions is True")
```

None of the conditions is True

Example 7

Bank transfer experience

```
amount_in_bank = 500

transfer_amount = 600

if (amount_in_bank > transfer_amount):
    print("Transfer Successful" )
else:
    print("Transfer Failed! Not enough funds")
```

Transfer Failed! Not enough funds

3.1.2 if, elif, else statement

Now let's imagine we have multiple conditions to check before the final **else** statement, this is where we can use the **elif** keyword to check for as many individual conditions as possible.

Example 1

```
x = 2
y = 6
if x == y:
    print('First condition True')
```

```
elif x > y :  
    print("Second condition True")  
elif x == 100:  
    print("Third condition True")  
else:  
    print("None of the above conditions are True")
```

None of the above conditions are True

Example 2

```
x = 2  
y = 6  
if x == y:  
    print('First condition True')  
elif y > x :  
    print("Second condition True")  
elif x == 100:  
    print("Third condition True")  
else:  
    print("None of the above conditions are True")
```

Second condition True

Since $y > x$, therefore, this condition will be executed

Example 3

Bank transfer experience

```
amount_in_bank = 600  
  
transfer_amount = 500  
  
if amount_in_bank > transfer_amount :  
    print("Transfer Successful" )
```

```
elif amount_in_bank == transfer_amount :  
    print("Transfer successful! Remember to add more fund next time")  
else:  
    print("Transfer Failed! Not Enough Funds")
```

Transfer Successful

Class Activity 13



What will get printed in the conditional statement below:

```
a = 3  
b = 5  
c = 8  
  
if a > b:  
    print("a is greater than b")  
elif b > a:  
  
    print("b is greater than a")  
else:  
    print("c is greater")
```

3.2 Python Loops

There are two types of loops in Python

- for loop
- and
- while loop.

3.2.1 for loop

A for loop is used to iterate over a sequence of object such as list, tuple, set, string or a dictionary.

For loop syntax

for *item* in object:

statements to do

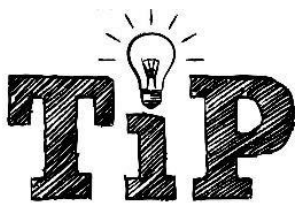
The naming of the **item** is completely up to you, so use your best judgment for choosing a name that makes sense and you will be able to understand when revisiting your code.

Example 1

With the for loop we can execute a set of statements, once for each item in a set, list, tuple, set etc. Consider this example that print each fruit in the fridge:

```
fridge = ["Mango", "Orange", "Apple", "Lemon", "Banana", "Grape", "Cherry", "Avocado", "Watermelon"]  
  
for fruit in fridge:  
    print(fruit)
```

```
Mango  
Orange  
Apple  
Lemon  
Banana  
Grape  
Cherry  
Avocado  
Watermelon
```



Code indentation becomes very important as we begin to work with loops and control flow.

Example 2

Printing the square of each number in the list

```
mylist = [1, 2, 3, 4]  
  
for number in mylist:  
    print(number**2)
```

1
4
9
16

As you can see, each of the number in the list has been squared with for loop

Example 3

Print the square of each of the number from 0 to 9 using range function

```
for i in range(10):  
    print(i, ".", i**2)
```

```
0 : 0  
1 : 1  
2 : 4  
3 : 9  
4 : 16  
5 : 25  
6 : 36  
7 : 49  
8 : 64  
9 : 81
```

Example 4

for loop with strings

```
country = "South Sudan"  
for character in country:  
    print(character)
```

```
S  
o  
u  
t  
h  
  
S  
u  
d  
a  
n
```

As you can see, each character in the South Sudan is being printed.

Example 4

for loop with tuple

```
teachers_name = ("Diric", "Bilen", "Baruk", "Jamal", "Gelila")  
  
for teacher in teachers_name:  
    print(f"This is {teacher}")
```

```
This is Diric  
This is Bilen  
This is Baruk  
This is Jamal  
This is Gelila
```

Example 5

for loop with a dictionary

```
life_expectancy = {"Nigeria": 60, "Kenya": 69, "Uganda": 68, "Ethiopia": 68, "Sudan": 67,  
                  "Rwanda": 65, "Tanzania": 64, "Somalia": 54}
```

Remember that dictionary is a key to value pairs

```
for country in life_expectancy.keys():  
    print(country)
```

```
Nigeria  
Kenya  
Uganda  
Ethiopia  
Sudan  
Rwanda  
Tanzania  
Somalia
```

```
for year in life_expectancy.values():  
    print(year)
```

60
69
68
68
67
65
64
54

```
for country in life_expectancy.keys():
```

```
    print(country)
```

```
    print(life_expectancy[country])
```

```
    print('\n')
```

Nigeria
60

Kenya
69

Uganda
68

Ethiopia
68

Sudan
67

Rwanda
65

Tanzania
64

Somalia
54

Introduction to continue and break statements in Python

We can use break or continue statement to alter the flow of a normal loop in Python. As you know, loop iterates over a block of code until the test expression is False. Sometime, we may wish to terminate or stop the current iteration or even the whole loop without checking test expression. We therefore, use **break** and **continue** statements in these cases.

The continue Statement

With the continue statement, we can stop the current iteration of the loop, and continue with the next item.

Example 1

Do not print Orange

```
fruits_list = ["Mango", "Orange", "Apple", "Lemon", "Banana", "Grape", "Cherry", "Avocado",  
"Watermelon"]  
for fruit in fruits_list:  
    if fruit == "Orange":  
        continue  
    print(fruit)
```

```
Mango  
Apple  
Lemon  
Banana  
Grape  
Cherry  
Avocado  
Watermelon
```

Orange is not among the fruits that were printed.

Example 2

A score that equals 50 will not be printed

```
score_list = [20, 30, 40, 50, 60]
```

```
for score in score_list:  
    if score == 50:  
        continue  
    print("Your score is", score)
```

```
Your score is 20  
Your score is 30  
Your score is 40  
Your score is 60
```

Example 3

A farmer in Tigray region of Ethiopia wants to know the list of good eggs in his poultry. An egg is good if it weighs above 60 grams (g). Use the for loop and continue statement to print out the good eggs.

```
egg_weight = [59, 56, 61, 68, 52, 53, 69, 54, 57, 51]
```

```
for egg in egg_weight:  
    if egg <= 60:  
        continue  
    print(egg)
```

```
61  
68  
69
```

You can see that only eggs that weigh more 60 grams (g) are printed out.

Example 4

```
for letter in 'code':  
    if letter == 'e':
```

```
continue
```

```
print('Current Letter is:', letter)
```

Current Letter is: c

Current Letter is: o

Current Letter is: d

Practice Question 3



Use the continue statement inside the for loop to skip the character from printing if the character is **o** in the string “Ethiopia”.

Answer to Practice Question 3

```
# Program to show the use of continue statement inside loops
```

```
for val in "Ethiopia":
```

```
    if val == "o":
```

```
        continue
```

```
    print(val)
```

E

t

h

i

p

i

a

Or use

```
country = "Ethiopia"

for i in country:
    if i == "o":
        continue
    print(i)
```

E

t

h

i

p

i

a

The break Statement

With the break statement, we can stop the loop before it has looped through all the items

Example 1

Exit the loop when fruit is “Grape”:

```
fridge = ["Mango", "Orange", "Apple", "Lemon", "Banana", "Grape", "Cherry", "Avocado", "Watermelon"]

for fruit in fridge:
    print(fruit)
    if fruit == "Banana":
        break
```


Mango

Orange

Apple

Lemon

Banana

Example 2

Exit the loop when fruit is “Grape”, but this time the break comes before the print:

```
fridge = ["Mango", "Orange", "Apple", "Banana", "Cherry",]  
for fruit in fridge:  
    if fruit == "Banana":  
        break  
    print(fruit)
```

Mango

Orange

Apple

Practice Question 4



Use the break statement inside the for loop to stop the character from printing if the character is **o** in the string “Ethiopia”.

Answer to Practice Question 4

```
for val in "Ethiopia":  
    if val == "o":
```

```
break  
print(val)
```

E

t

h

i

As you can see, when the character is **o**, the program stops from printing the next character.

Example 3

Loop through the list **numbers** and print out all even numbers from the numbers list in the same order they are received. Don't print any numbers that come after 35 in the sequence.

```
numbers = [751, 202, 784, 451, 160, 131, 208, 119, 401, 285, 780, 307, 525, 347, 344, 415, 117,  
35, 59, 301, 63, 417, 665, 375, 19, 190, 784, 392, 36, 95, 742, 741, 186, 262, 153, 218, 707, 1  
44, 36, 175, 623, 366, 397, 778, 128, 415, 753, 145]
```

```
for number in numbers:
```

```
    if number % 2 == 0:
```

```
        print(number)
```

```
    if number == 35:
```

```
        break
```

```
    print(number)
```

202

784

160

208

780

344

Example 4

Loop through the list **numbers** and print out all even numbers from the numbers list in the same order they are received.

```
numbers = [751, 202, 784, 451, 160, 131, 208, 119, 401, 285, 780, 307, 525, 347, 344, 415, 117,  
35, 59, 301, 63, 417, 665, 375, 19, 190, 784, 392, 36, 95, 742, 741, 186, 262, 153, 218, 707, 1  
44, 36, 175, 623, 366, 397, 778, 128, 415, 753, 145]
```

```
for number in numbers:  
    if number % 2 == 0:  
        print(number)
```

202

784

160

208

780

344

190

784

392

36

742

186

262

218

144

36

366

778

128

3.2.2 The while loop

A while loop will repeatedly execute a single statement or group of statements as long as the condition being checked is True. The reason it is called a ‘while loop’ is because the code statements are looped through over and over again until the condition is no longer True.

Syntax of while Loop

```
while test_expression: Body of while
```

Example 1

Print number as long as number is less than 10:

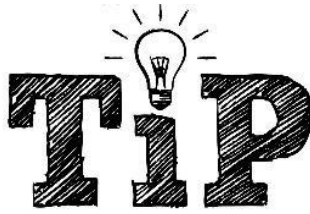
```
number = 1
while number < 10:
    print(number)
    number = number + 1
```

1
2
3
4
5
6
7
8
9

As you can see, it only prints number that is less than 10.



You must remember to increase number i.e. `number + 1`, or else the loop will continue forever.



One unique feature about the while loop is that it requires relevant variables to be ready, in this example we need to define an indexing variable, number, which we set to 1.

Example 2

We can also do the increment in this way:

```
number = 1

while number < 10:
    print(number)
    number += 1
```

1
2
3
4
5
6
7
8
9

Example 3

Printing number that is less than 20

```
a = 8

while a < 20:
```

```
print(f"{a} is less than 20")  
a = a + 1
```

```
8 is less than 20  
9 is less than 20  
10 is less than 20  
11 is less than 20  
12 is less than 20  
13 is less than 20  
14 is less than 20  
15 is less than 20  
16 is less than 20  
17 is less than 20  
18 is less than 20  
19 is less than 20
```

Example 4

Printing number that is less than 20

```
# Start by setting variable x to 0  
x = 0  
while x < 5:  
    print('x is currently')  
    print(x)  
    print("\nAdding 1 to x") # \n is to print to a new line  
    x = x + 1 # alternatively you could write x += 1
```

```
x is currently  
0
```

```
Adding 1 to x  
x is currently  
1
```

```
Adding 1 to x  
x is currently  
2
```

```
Adding 1 to x  
x is currently  
3
```

```
Adding 1 to x  
x is currently  
4
```

```
Adding 1 to x
```



Be careful with the while loop! There is a potential to write a condition that always remains True. That is, you have an infinite while loop. If this happens to you, you can stop/restart the kernel.

The continue Statement

With the continue statement we can stop the current operation or iterator, and continue with the next:

Example 1

Continue to the next iteration if i is 4:

```
i = 0  
while i < 10:  
    i += 1  
    if i == 4:
```

```
continue
```

```
print(i)
```

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

Example 2

Continue to the next iteration if number is 7:

```
number = 0  
while number < 10:  
    number += 1  
    if number == 7:  
        continue  
    print(number)
```

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

The break Statement

With the break statement we can stop the loop even if the while condition is `True`.

Example 1

Exit the while loop when i is 10:


```
i = 1
while i < 15:
    print(i)
    if i == 10:
        break
    i += 1
```

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

Example 2

Exit the while loop when x is 3:

```
x = 0
while x < 10:
    print(x)
    if x == 3:
        break
    x = x + 1
```

```
0
1
2
3
```

Example 3

Exit the while loop when x > 12:

```
x = 0
while x < 20:
    print(x)
```

```
if x > 12:
```

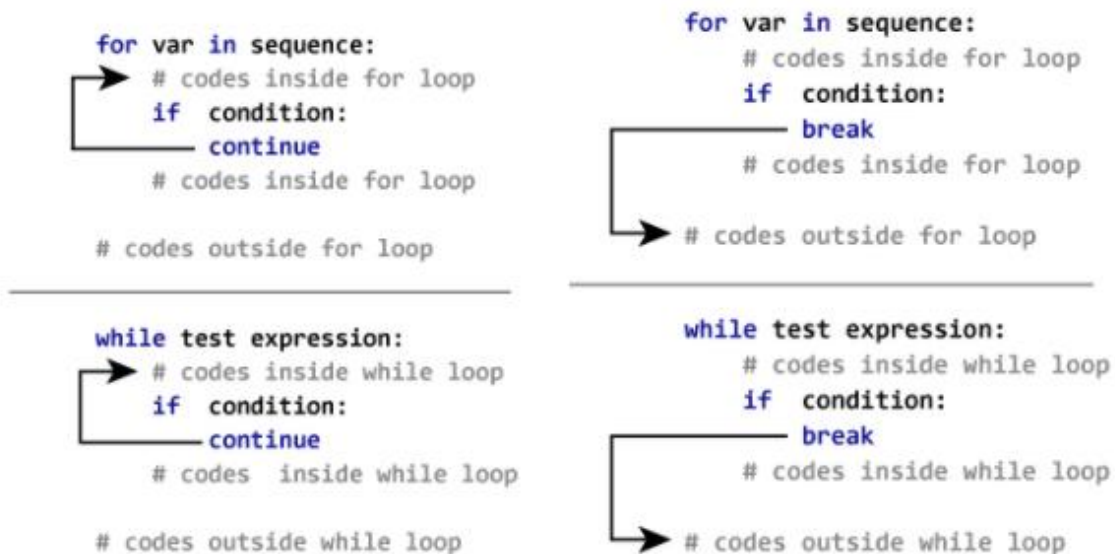
```
    break
```

```
x = x + 1
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
```

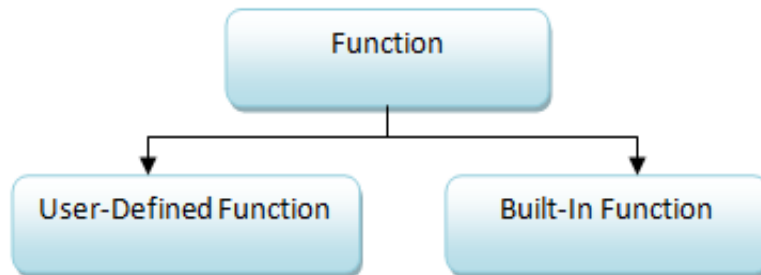
General tips for break and continue statements in for and while loops

The `break` statement is used to exit a `for` loop or a `while` loop, whereas `continue` is used to skip the current block, and return to the `for` or `while` statement.



3.3 Python functions

Python function is a block of organized and reusable code that is used to perform a single and related action. It is a piece of code that runs when it is called or referenced. Python provides many inbuilt functions like `print()`, `input()`, `type()`, `len()`, etc. but it also gives freedom to create your own functions.



3.3.1 Types of Functions

Function extends the functionality of Python. Basically, we can categorize functions into the following two types:

Built-in functions - Functions that are pre-defined in Python

User-defined functions - Functions defined by the users themselves

Advantages of Python Functions

Function allows us to automate repetitive tasks in a more powerful way than copy-and-pasting.

Writing a function has the following advantages:

- ✚ You can give a function a name that makes your code easier to understand and remember.
- ✚ As something changes, you only need to update code in one place, instead of many places.
- ✚ it avoids repetition and makes the code reusable.
- ✚ Improves maintainability of the code.

When should you write a function?

You should consider writing a function whenever you have copied and pasted a block of code more than twice.

3.3.2 Creating a function in Python

You can define functions by following the rules below:

- Function blocks begin with the keyword **def** followed by the **function_name**, parentheses (**()**), a colon (**:**), and is indented. Function name follows the same rules of naming variables in Python.
- Any parameters or arguments should be placed within these parentheses. You can add as many arguments as you want, just separate them with a comma. Parameters (arguments) helps to pass values to a function. They may be optional sometimes.
- A colon (**:**) to mark the end of the function header.
- Optional documentation string (docstring) to describe what the function does (for documentation purposes).
- One or more valid python statements that make up the function body. Statements must have the same indentation level (Jupyter lab/notebook does this for you when you press return button after the colon mark).
- Functions may return a value to the caller, using **return** statement. This is optional.

Function syntax

```
def functionname(parameters):  
    "function_docstring describing what the function does"  
  
    # list of expressions to be executed  
  
    return (expression)
```

In graphical form, here is a function that returns the sum of two given numbers

1. def keyword 2. function name 3. function arguments inside ()

```
def add(x, y):
    print(f'arguments are {x} and {y}')
    return(x + y)
```

4. colon ends the function definition

5. function code 6. function return statement



The terms parameter and argument can be used for the same thing: information that are passed into a function.

3.3.3 Calling a Function in Python

You can call a function by its name. If the function accepts parameters or arguments, you will have to pass them while calling the function.

Example 1

```
def sum_two_numbers(a, b):
    "This function add two numbers together"
    return (a + b)
```

```
sum_two_numbers(9, 7)
```

16

Since function is reusable. We can also get the sum of 6 and 9 by using the `sum_two_numbers()` function

```
sum_two_numbers(6, 9)
```

15

3.3.4 Indentation error

If your code is not well indented inside a function, it will throw an error. For example,

```
def sum_two_numbers(a, b):  
    "This function add two numbers together"  
y = a + b # Not well indented  
    return (y)
```

```
File "<ipython-input-45-ed23e87d8ece>", line 4  
    return (y)  
    ^  
IndentationError: unexpected indent
```

Example 2

The following function takes two numbers as arguments and prints out their sum:

```
def add(x, y):  
    "This function add two numbers together in another form"  
    print(f"The sum of {x} and {y} is {x+y}")  
  
add(2, 6)
```

The sum of 2 and 6 is 8

You may also specify function arguments and supply their values

```
add(x = 10, y = 32)
```

The sum of 10 and 32 is 42

3.3.5 Default Arguments in a Function

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument. The following example gives an idea on default arguments, it prints default age if it is not passed.

```
def printinfo(age=25, gender = "Female"):  
    "This prints the info about a person"  
    print(f"You are a {gender} and your age is {age} years")  
  
printinfo()
```

You are a Female and your age is 25 years

```
printinfo(age= 19)
```

You are a Female and your age is 19 years

```
printinfo(age = 20, gender= "Male")
```

You are a Male and your age is 20 years

Example 3

This function prints out the demographic information about a student taken a particular course

```
def student_info(name, age, gender, course_code, country):  
    print(f"Information about a student taken {course_code} course:")  
    print ("Name:", name)  
    print ("Age:", age)  
    print ("Gender:", gender)  
    print("Country:", country)  
  
# Now you can call student_info function  
student_info(name = "Zula", age = 18, gender = "Female", course_code = "CS 22", country= "Ethiopia")
```

Information about a student taken CS 22 course:

Name: Zula

Age: 18

Gender: Female

Country: Ethiopia

```
student_info(name = "Safari", age = 21, gender = "Male", course_code = "CS 21", country= "Kenya")
```

Information about a student taken CS 21 course:

Name: Safari

Age: 21

Gender: Male

Country: Kenya

Example 4

Your function can also take user input. For example:

```
number = int(input("Please enter a number to check for odd or even"))

def odd_even_checker(number):
    if number % 2 == 0:
        print(f"{number} is an even number")
    else:
        print(f"{number} is an odd number")

odd_even_checker(number)
```

Please enter a number to check for odd or even

13 is an odd number



A function can have multiple return statements. However, when one of the return statements is **True**, the function execution will terminate and the value is returned to the caller.

Example 5

```
def even_odd_checker(number):
    if number % 2 == 0:
```



```
    return("Even")
else:
    return("Odd")

even_odd_checker(14)
```

‘Even’

```
even_odd_checker(number= 19)
```

‘Odd’

Example 6

This task was taken from DataCamp!

Add a function named `list_benefits()` that returns the following list of strings: “More organized code”, “More readable code”, “Easier code reuse”, “Allowing programmers to share and connect code together”

```
list_of_strings = ["More organized code", "More readable code", "Easier code reuse",
                  "Allowing programmers to share and connect code together"]

def list_benefits():
    for string in list_of_strings:
        print(string) # If you use return statement instead of print, only "More organised code" will
be returned

list_benefits()
```

More organized code

More readable code

Easier code reuse

Allowing programmers to share and connect code together

Alternatively, we can write it in this way:

```
def list_benefits(benefit):  
    for i in benefit:  
        print(i)  
  
list_of_strings = ["More organized code", "More readable code", "Easier code reuse",  
                  "Allowing programmers to share and connect code together"]  
  
list_benefits(list_of_strings)
```

More organized code

More readable code

Easier code reuse

Allowing programmers to share and connect code together

Class activity 16 (Peer to peer review activity)

Peer to Peer Interaction



Visit the LMS, locate forum activity and participate in the discussion

- ✚ Write a function that asks the user to enter two numbers. This function will print their sum and their difference each on a single line.
- ✚ Write a function that prompts the user to enter an integer `n` and outputs the letter “x”, `n` times on a single line, without a space.

- ✚ Write a function to convert to Fahrenheit a temperature given in degree Celsius.


You can use this formula:

$$T_c * 1.8 + 32$$

where `Tc` is the Celsius degree to be converted.

Use your function to convert the following Celsius to Fahrenheit

 100

 75

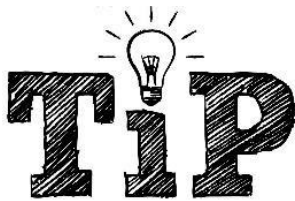
 120

3.3.6 Anonymous function

Anonymous functions don't have a name and is not declared in the standard manner of a function by using the `def` keyword. We can define anonymous function in Python using `lambda` keyword.

Syntax

`lambda` arguments : expression



Anonymous function is also known as `lambda` function. The expression in the `lambda` function is executed and the result is returned

Example 1

Write an anonymous function to find the square of a given number

```
square = lambda x : x ** 2
```

The argument of this function is `x` while `x ** 2` is the expression that we want to evaluate. A lambda function can take any number of arguments, but can only have one expression. You can call an anonymous function by its name and supply values to its arguments.

```
square(x = 10)
```

100

```
square(8)
```

64

Example 2

Multiply two numbers together and return the result:

```
product = lambda x, y : x * y
```

```
product(2, 6)
```

12

```
product(2, 8)
```

16

Class Activity 17



Create a lambda function to calculate the difference between two numbers.

Use your lambda function to calculate the sum of

- -18 and 20
- 19 and 21
- 16 and 5

Additional resources

For more resources in this section please consider the following:

 <https://www.guru99.com/functions-in-python.html>

3.4 Introduction to Python libraries

A library is a collection of Python functions that are written in Modules to extend basic Python functionality. A library can contain a set of functions relating to a specific topic or tasks. For example, Pandas for data manipulation and analysis, NumPy for scientific computing and manipulation n-dimensional arrays, Matplotlib and Seaborn for data visualization while scikit-learn is for building machine learning models.



Module is a file which contains various Python functions with the **.py** extension file which has python executable code.

Package is a collection of modules.

Library is a collection of packages.

3.4.1 How to import Python package

Before you can use any function in a package, you will need to import the module that has that function. To import a module, simply type

```
import module_name
```

For example, if I want to import a **NumPy** module, I will type

```
import numpy
```

The following statement allows us to use a module attribute (or function).

```
module_name.attribute
```

This means that we will have to refer to the function in dot notation. For example:

module_name.function

The **NumPy** module provides various scientific functions e.g. trigonometric functions, exponent, logarithm, and mathematical constant.

```
import numpy
```

We can use the module in the following ways:

Example 1

For the constant π (**pi**) which is rounded to 3.142

```
numpy.pi
```

```
3.141592653589793
```

Example 2

For the square root of 16 ($\sqrt{16}$)

```
numpy.sqrt(16)
```

```
4.0
```

To load selected functions from a module, we use the following syntax

```
from module_name import funcname
```

For example, import only **sqrt**, **pi**, **floor** from **numpy** module

```
from numpy import sqrt, pi, floor
```

You can then use those functions without referencing numpy again.

Example 1

```
sqrt(64)
```

```
8.0
```

Example 2

```
pi
```

```
3.141592653589793
```

Example 3

```
floor(3.141592653589793) # This function only returns the integer part.
```

```
3.0
```

3.4.2 Importing packages with alias

It is common for Python users to import packages with some alias. Alias helps us to shorten the name of the packages while referencing a certain function with it.

The syntax looks like this:

```
import module_name as another_name
```

For example,

```
import numpy as np
```

We can now refer to the function as `np.function_name` rather than `numpy.function_name`

Example 1

For the constant π (**pi**) which is rounded to 3.142

```
np.pi
```

```
3.141592653589793
```

Example 2

For the square root of 16 ($\sqrt{16}$)

```
np.sqrt(16)
```

```
4.0
```

For some other modules, it is common to use aliases. for example, consider the following data science packages:

 Pandas

 Matplotlib.pyplot

 Seaborn

We can import them as follows:

```
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns
```

We have come to the end of the introduction to Python programming (CS 2 1). In the next course, which is Introduction to Data Science I (CS2 2), you will learn everything about data science. I hope to see you there. Best and have fun!

Class activity 18 (Peer to peer review activity)

Peer to Peer Interaction

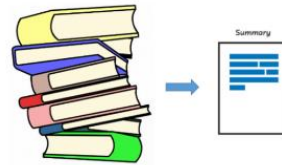


Visit the LMS, locate forum activity and participate in the discussion

+ Search for any other packages in Python that we have not used in this course.

+ Import them.

Summary of Study Unit 3



In this study unit, you have learnt that:

1. Conditional statement in Python includes **if else** statement
2. Python has two loops, the for **loop** and the **while** loop
3. **continue** and **break** statements can be used in both **for** loop and **while** loop
4. Python has built-in functions and user-defined functions
5. Built-in functions in Python includes **print()**, **input()**, and **int()**.
6. An anonymous function always has the syntax **lambda arguments: expression**
7. Python libraries can be imported by using **import** keyword