Contents

[Table of Figures 1](#_Toc116212011)

[Flow Control 3](#_Toc116212012)

[Code 4](#_Toc116212013)

[Loops 5](#_Toc116212014)

[For Loops 5](#_Toc116212015)

[IF THEN ELSE 7](#_Toc116212016)

[Python break, continue, pass statements 8](#_Toc116212017)

[Nested tuples in lists 9](#_Toc116212018)

[Tuple Unpacking 10](#_Toc116212019)

[Range Function 10](#_Toc116212020)

[Iterating through a dictionary – Exercise 4 11](#_Toc116212021)

[While loops 12](#_Toc116212022)

[Code 12](#_Toc116212023)

[References 15](#_Toc116212024)

# Table of Figures

[Figure 1 - Result of if statement 3](#_Toc116211997)

[Figure 2 - Experimenting with the values 3](#_Toc116211998)

[Figure 3- If then else statement 4](#_Toc116211999)

[Figure 4 - Testing Loops - my\_for.py 5](#_Toc116212000)

[Figure 5- Printing odd and even numbers - my\_for.py 6](#_Toc116212001)

[Figure 6 - For loop with print indented - my\_for2.py 6](#_Toc116212002)

[Figure 7 - For loop with the print not indented - my\_for2.py 7](#_Toc116212003)

[Figure 8 - Looking for the letter "K" within the string - my\_for3.py 7](#_Toc116212004)

[Figure 9 - Looking for a character, not in the string - my\_for3.py 8](#_Toc116212005)

[Figure 10- Results of control statements - pass, continue and break – my\_for4.py 9](#_Toc116212006)

[Figure 11 - Nesting a tuple in a list 9](#_Toc116212007)

[Figure 12 - Tuple Unpacking 10](#_Toc116212008)

[Figure 13 - Tuple range function 11](#_Toc116212009)

[Figure 14 - Iterating through a dictionary 11](#_Toc116212010)

# 

# Flow Control

In programming, there are times when we must make decisions and then carry out the subsequent block of code in accordance with those decisions. Statements such as if, if-else, and nested-if as used in control flow coding [1]. In Python, control flow is determined by colons (:) and indentation.

Graphical user interface, text, application

Description automatically generated

Figure - Result of if statement

Graphical user interface, text, application

Description automatically generated

Figure - Experimenting with the values

An if then else statement to see if three is less than 2 and 5 less than nine was run to see if it was true or false. Results are shown in figure 2 below.

Graphical user interface, text, application, Word

Description automatically generated

Figure - If then else statement

## Code

a=False

b=False

c=False

if a:

print("a was True")

elif b:

print("b was True")

elif c:

print("c was True")

else:

print("None of our Booleen variables were True")

if (3<2) and (5<9):

print("Yup!")

else:

print("Nope!")

# Loops

The following types of loops are available in the Python programming language to satisfy looping needs. Python offers three options for running the loops. The core functionality of all the methods is the same, although their syntax and time required for condition checking vary.

* While Loop
* For Loop
* Nested Loops [2].

## For Loops

Graphical user interface, text, application, Word

Description automatically generated

Figure - Testing Loops - my\_for.py

I examined this code:

iterable\_variable = [1,2,3,4,5,6] for item in iterable\_variable:

if item %2 != 0:

print(item)

The exclamation mark means not equal.

%2 Modulo Operator – Helps find even or odd numbers

!= 0: - Not equal to 0

In figure 5 below, the code shown will allow the programme to print odd and even numbers from the loop set:

Graphical user interface, text, application

Description automatically generated

Figure - Printing odd and even numbers - my\_for.py

Indentation is significant when it comes to coding in Python. Looking at the code below in figure 6, the final line of the code (Print(total), when indented, will run right through and print all the results of the **for loop**.

Graphical user interface, text, application

Description automatically generated

Figure - For loop with print indented - my\_for2.py

The **For loop** with **print** indented Looking at the code below in figure 7, the final line of the code (Print(total), is not indented; it will run through all the options and only print the final result.

Graphical user interface, text, application, Word

Description automatically generated

Figure - For loop with the print not indented - my\_for2.py

## IF THEN ELSE

Once the If Them Else exercise was completed, it was noted that the code would run through the entire string to find the relevant character or not. Figure 8 shows the code and result where the If Then Else statement looks through the name to find the character.

Graphical user interface, text, application

Description automatically generated

Figure - Looking for the letter "K" within the string - my\_for3.py

Figure 9 shows the code and result when a letter not in the string is searched for. The code runs through all the characters in the string and eliminates all the not-searched-for characters.

Graphical user interface, text, application

Description automatically generated

Figure - Looking for a character, not in the string - my\_for3.py

## Python break, continue, pass statements

The loop control statements interrupt the execution flow and stop/skip the iteration as necessary. The loop's regular method can be altered by using Python's break and continue commands. The break statement handles ending the loop in which it is used. The code that follows the continue statement is skipped, and control is returned to the start for the subsequent iteration. Inside loops, functions, classes, and if-statements that are intended to be implemented later, Python pass statements are used as placeholders [3].

The result of using these control statements in the given exercise is shown in figure 10 below. It will pass if number one is found; if number 2 is found, it will loop. If number three is found, it will print a statement.

Graphical user interface, text, application, Word

Description automatically generated

Figure - Results of control statements - pass, continue and break – my\_for4.py

## Nested tuples in lists

Tuples can contain other compound objects, including lists, dictionaries, and other tuples. Hence, tuples can be nested inside of other tuples or lists. [4]. Figure 11 below shows the code and output of nesting a tuple in a list.

Text

Description automatically generated

Figure - Nesting a tuple in a list

## Tuple Unpacking

Usually, tuples include a series of heterogeneous elements that can be retrieved by unpacking or indexing. Tuples are immutable (or even by attribute in the case of named tuples) [5]. Figure 12 below displays some code and output of unpacking a tuple.

Text

Description automatically generated

Figure - Tuple Unpacking

## Range Function

The range function in Python can be used to create a range of numbers from a starting point to an endpoint [6]. The syntax for this function is range(start, stop, step), **start**: the integer starting from which the sequence of integers is to be returned, **stop**: integer before which the sequence of integers is to be returned. The range of integers ends at stop – 1. **step**: integer value which determines the increment between each integer in the sequence. Figure 13 below shows the use of this code and the results. In this, we are asking the system to return the range 1 (start) to 100 (stop) in increments of 5 (step).

Text

Description automatically generated with medium confidence

Figure - Tuple range function

## Iterating through a dictionary – Exercise 4

The simplest way to iterate is through a dictionary in Python. Just put it directly into a **"for"** loop. Using code **for the key in the scan: print(key)** will return a list of all keys in the dictionary but not the values. This code **for key in scan.items(): print(key)** will print all keys and their corresponding values. Figure 14 below shows the use of this code and the corresponding result.

Text

Description automatically generated

Figure - Iterating through a dictionary

## Code

iterable\_variable=[1,2,3,4,5,6]

for item in iterable\_variable:

#for each item, execute this code block

print(item)

**print("Print all odd numbers from the loop")**

for item in iterable\_variable:

if item%2!=0:

print(item)

**print("Print all even numbers from the loop")**

for item in iterable\_variable:

if item%2!=1:

print(item)

iterable\_variable=[1,2,3,4,5,6]

total=0

**print("With the Print command indented the for loop processes all the options and then prints them")**

for item in iterable\_variable:

total=total+item

print(total)

**print("With the Print command is not indented the for loop processed all the options and then prints the final result")**

for item in iterable\_variable:

total=total+item

print(total)

**# Define a string to iterate over**

for this\_letter in "Karen McDonnell":

#Specify which letter to test for

if this\_letter == "p":

#Found the test letter

print(f"Woo hoo, found a {this\_letter}!")

#Exit the current loop

break

else:

#Didn't find the test letter

print(f"Aww man, I didn't want a {this\_letter}!")

**print("Pass, Continue, Break")**

my\_list=[1,2,3,0]

for my\_number in my\_list:

if my\_number==1:

pass

if my\_number==2:

continue

if my\_number==3:

print(f"Found the number {my\_number}")

if my\_number==0:

break

**print("nest tuples in lists")**

list\_of\_tuples = [(1,2), (3,4), ("A", "B")]

for item in list\_of\_tuples:

print(item)

**print("Tuple unpacking")**

list\_of\_tuples = [(1,2), (3,4), ("A", "B")]

for a,b in list\_of\_tuples:

print(a)

print(b)

**print("Tuple Indexing")**

for index in range(1, 100, 5):

print(index)

**Iterating through a dictionary – Exercise 4**

scan = {"192.168.3.10": "80", "192.168.3.11": "443", "192.168.3.55": "22"}

#Using code for the key in the scan: print(key) will

#return a list of all keys in the dictionary but not the values.

for key in scan:

print(key)

#Will return all keys and values

for key in scan.items():

print(key)

# While loops

Until a specified condition is met, a block of statements will be periodically executed in a Python while loop. Furthermore, the line in the program that follows the loop is run when the condition changes to false [7]. Figure 15 below displays an example of Python's code required to set a while loop. This loop will run until it equals ten and then stops, adding x + 1 as it goes.

Text

Description automatically generated

Figure - While Loop

## Code

x = 0

while x < 10:

print(f"X is = {x}")

x = x + 1

else:

print(f"As x is now = {x}, we are all finished")

# List Comprehensions

A considerably shorter syntax is available in Python List comprehension for building a new list from the values of an existing list [8].

# Exercise 4b

This exercise requires a conversion of Kelvin to Celcius and Fahrenheit. Figure 16 displays the code below.

# A screenshot of a computer Description automatically generated

Figure - Exercise 4b - Kevin to Celcius and Fahrenheit conversion

# 

# References

[1] D. Agarwal, “Control Flow in Python,” *GeeksforGeeks*, Dec. 27, 2017. https://www.geeksforgeeks.org/python-if-else/ (accessed Oct. 06, 2022).

[2] GeeksforGeeks, “loops in python,” *GeeksforGeeks*, Jun. 07, 2017. https://www.geeksforgeeks.org/loops-in-python/ (accessed Oct. 06, 2022).

[3] S. Campbell, “Python break, continue, pass statements with Examples,” Mar. 01, 2020. https://www.guru99.com/python-break-continue-pass.html (accessed Oct. 06, 2022).

[4] Baldwin, By, and Baldwin, “Learn to Program using Python: Nested Tuples,” *Developer.com*, May 06, 2004. https://www.developer.com/languages/learn-to-program-using-python-nested-tuples/ (accessed Oct. 07, 2022).

[5] GeeksforGeeks, “Python Tuples,” *GeeksforGeeks*, Mar. 13, 2018. https://www.geeksforgeeks.org/python-tuples/ (accessed Oct. 09, 2022).

[6] artturijalli, “How to Use range() Function in Python,” *codingem.com*, Nov. 22, 2021. https://www.codingem.com/how-to-use-range-function-in-python/ (accessed Oct. 09, 2022).

[7] nikhilaggarwal3, “Python While Loop,” *GeeksforGeeks*, Nov. 11, 2019. https://www.geeksforgeeks.org/python-while-loop/ (accessed Oct. 09, 2022).

[8] riturajsaha, “Python - List Comprehension,” *GeeksforGeeks*, Oct. 28, 2020. https://www.geeksforgeeks.org/python-list-comprehension/ (accessed Oct. 09, 2022).