Software for Digital Innovation (CIS4044-N)	
Name	Fatimah Oladejobi
Student Number	B1421693
Portfolio No.	2 – Selection and Iteration

Describe the following programming terms	
Selection	Selection is a statement used to choose/select part of a program to be executed based on logical condition. **Notable Points Regarding Selection** • Selection enhances frontend user experience considering the executable options/condions available. • Selection is done using 'if' statements where conditions are evaluated. • If the condition(s) is met (i.e. True), the specified action in the body of the statement is performed. Otherwise, If the condition is not met (i.e. False), the instruction in the "if" statement is skipped. • Additional selection statements can be added to control the program flow. These include multiple 'elif' (else if) and a single 'else'.

```
#Fatimah Oladejobi,
#17/11/2021
#Program on eligibility to vote
name = input("Enter your name: ")
age = int(input("Enter your age: "))
citizenship = input("Are you UK or Britain? enter yes
or no: ")

if age < 18:
    print(name, "oh sorry! you're ineligible to
vote")
elif age > 18 and citizenship == "UK or Britain":
    print(name, "you are eligible to vote")
else:
    print("sorry", name, "you are ineligible to
vote")

#Possible response
#Enter your name:  Fatimah
Enter your age:  19
Are you UK or Britain? enter yes or no:  no
sorry Fatimah you are ineligible to vote
```

Relational operators

Write your description here.

Are used to compare numeric, character string, or logical data with the result(return) either true (1) or false (0). The result can be used to decide program flow.

Operators	Relation	Examples
LT or <	Less-than	4 < 5
GT or >	Greater-than	5 > 4
LE or <= or =< or #<	Less-than or equal to	4<=5
GE or >= or => or #>	Greater-than or equal to	5>=4

```
#Fatimah Oladejobi,
#17/11/2021
# Provide a Python code example here...
#Comparing two different ages
# Request user to input two ages.
first_age = input("Enter the first age: ")
second_age = input("Enter the second age: ")

# Cast ages to integer.
first_age = int(first_age)
second_age = int(second_age)

# Compare ages.
if first_age > second_age:
    print(first_age, "is bigger than", second_age)
elif second_age > first_age:
    print(second_age, "is bigger than", first_age)
else:
    print("Both are equal")

# Subsequent line of code...
print("Thank you!")
```

Boolean operators

Boolean Operators are operators that are used to handle multiple conditions in if statement. Hence, they're used to evaluate decisions that are based on more than one factors. They include and(conjunction), or(disjunction), not(negation). These Boolean operators feature in groups of statements/conditions (known as compound statements/conditions).

In practice, 1 is evaluated as True, while 0 is evaluated as False. All non-zero numbers and all non-empty strings are True and zero and the empty string ("") is False.

```
#Fatimah Oladejobi,
#17/11/2021
#Example of Boolean operation
x = 10  #True
y = 0   #False
print(x and y)#This prints out 0 because one of the
#conditions is “and” condition. True and False = 0
print(x or y)#This prints out 10 due to the or
#condition
print(not x)# False

#For Compound condition
#To compare group of numbers
a = 200
b = 33
c = 500
if a > b or a > c:    #one of these operations must
be true
  print("At least one of the conditions is True")

# Ask the user for a number.
x = int(input("Enter a number"))

# INCORRECT: Final elif never executes.
if x > 100:
    print("x is greater than 100")
elif x % 2 == 0:      #elif x > 100 and x % 2 == 0
    print("x is even")
elif x > 100 and x % 2 == 0:
    print("x is both greater than 100 and even")
else:
    print("x is odd number")

#Note that the x variable is almost inevitable in
Boolean expressions
```

Boolean expression	Is an expression that is used to test conditions yielding just two outcomes: true or false.

```
#Fatimah Oladejobi,
#17/11/2021
#Example of Boolean Expression
is_hot = False
is_cold = True

if is_hot:
    print("It's a hot day!!")
    print("Drink plenty of water")
elif is_cold:
    print("It's a cold day!")
    print("wear warm clothes")
else:
    print("It's a lovely day!")

print("Enjoy your day!")
```

if, elif, else statements

Write your description here.

The if, elif, else statements are selection statements.

Notable points about the if, elif, else statements

- The **if** line of statement must execute first. If it exits.
- If the **elif** condition(s) is/are present, they are evaluated after the if statement.
- The else statement must be executed last, after the if and elif statement.
- There can be many elif statements between the if and **else**.
- When we define an if statement, the block of statement must be specified using proper and consistent indentation (a tab/four spaces).
- Once the first true condition is executed, the remaining statements will be disregarded and therefore not executed.
- If the block of statement is evaluated as True, it will execute the branch (the true block of statements) and if it is False, then the false block of statements is executed, before proceeding to the next line of code.
- The ‘elif’ and ‘else’ are optional conditions, proffering alternative set of instructions to be executed if the condition in the ‘if’ statement is evaluated as False.
- The ‘elif’ statement is often used to test multiple conditions.

The if, elif, else statements are often structured as follows:

```
if condition:

   If body

elif condition:

    elif_body

elif condition:

    second_elif_body

else:
```

	else_body

```
#Fatimah Oladejobi,
#17/11/2021
#Program to convert scores to grades.
while True:
    score = input("Please input your score: ")
    #Cast the score into integer
    score = int(score)
    if score == 0: # If statement must come first.
        print("Grade N/S")  # Perform this action, if
not proceed
                            # to the next selection
expression.
                            # No score
    elif score > 0 and score <= 39: # All the
optional elif statements follow,
                                    # Python
evaluates each in order.
        print("Grade F")
        print("Are you understanding the content?") #
Body can contain several expressions.
    elif score >39 and score <= 49:
        print("Grade D")
    elif score >49 and score <= 59:
        print("Grade C")
    elif score >59 and score <= 69:
        print("Grade B")
    elif score >69 and score <= 79:
        print("Grade A")
    elif score >79 and score <= 100:
        print("Grade A*")
        print("Great job!")
    else: # Optional else statement always goes /
execute last.
          # Python will perform these actions if all
others fail.
        print("Please input a number between 0-100.")



# Selection statements can be used in loops.
x = 0
n = input("Please input a number: ")
n = int(n) # Convert input to integer.
for i in range(n): # Iterate through the numbers 0 to
n exclusive.
    x += i # Add i to x.
if x >= 100:
```

	break # Stop the program if x is more than 100. print(x) # Print the value of x.
Iteration	Iteration has to do with repeating/controlling the number of times a section of code will be executed. One example is using a loop, explicit or implicit, to evaluate and perform an action on a group of items. **<u>Types of iteration</u>** • Definite iteration: entails taking each instance of a list or sequence, one after another, to repetitively execute the same block of code. The number of repetitions while executing a code block is explicitly defined in advance. Example is observed in **for loop**. • Indefinite iteration, the code block is executed until some condition is met. For example, using a while loop. **<u>In summary:</u>** • Iteration can be automatically executed or initiated, such as a **for loop**, a map, or a list comprehension, etc. • Iteration can be definite or indefinite in its execution. • Another common Iteration is Tail-recursive iteration. • The flow of Iteration can be controlled using the continue, pass or break statements.

	#Fatimah Oladejobi,
	#17/11/2021
	# Iterates over the numbers 10 to 100 with a step of 10.
	for x in range(10, 101, 10):
	# Prints multiples of 10 up to 100.
	    print(x)
	
	# Iterate over each number, to convert to positive int, and store in a new list variable.
	nums = [33.6, -210.1, 55.3, 28.4, -10.2, 77.9, 22.7]
	pos_int = [abs(int(num)) for num in nums]
	pos_int # *Prints list to console.*

for statement	For statement could iterate over the items of any sequence, such as a list or a string, tuple, set or dictionary I.e., a For statement iterates over the members of the sequence in order, executing the code block each time. **#Notable points about "for" statements** • A "for" loop can have an optional else block which is executed if the items in the sequence used in the for loop are exhausted. • Therefore, a "for" loop's else block is executed if no break occurs. • The range() function can be used to specify the number of times to loop through a code block. • The range() returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number (exclusive of the final number) as the case may be. • The break keyword can be used to stop a for loop, in which case, the else part is ignored.

```
#Fatimah Oladejobi,
#17/11/2021
#Possible values in a tossed coin
#import random
#Toss
coins = ['H','T']
dataList = []
for i in range(100):
    dataList.append(random.choice(coins)) # randomly
choice "H" or "T" and store the value in a list named
dataList
print(dataList)

#print out the list in 5 rows of 20.
start = 0
end = 20
for i in range(5):             # loop for row counter
    for j in range(start,end): # loop for 20 data in
a row
        print(dataList[j],end="  ")
    print("\n")                   # print new line

# Increase the value 20 in each loop to get the next
values
  start = start+20
  end = end+20
# Increase the value 20 in each loop to get next
values

#To count how many times three heads were tossed in a
row

counterHead = 0
c = 0
start = 0
end = 20
for i in range(5):             # loop for row counter
    for j in range(start,end): # loop for 20 data in
a row
        print(dataList[j],end="  ")
        if dataList[j] == 'H':
            c +=1
            if c == 3:
                counterHead += 1
                c = 0
        else:
            c = 0
    print("\tCounter Head (3 Consecutive Heads in a
Row): ", counterHead)

    print("\n")                     # prints new line


# increase the value 20 in each loop to get next the
values
    start = start+20
    end = end+20
# increase the value 20 in each loop to get the next
values
    counterHead = 0
    c = 0
```

while statement

While Statement repeatedly executes a target statement if the given condition (Boolean) is True.

#Notable Features of While Statement

- while loops can also (just like in for loop) have an optional else condition, to execute a block of code when the condition is no longer True.
- After each iteration, the conditional expression is checked to ascertain that it is still True.
- When the condition becomes false, program control passes to the next line (following the loop).
- The keyword: break can be used to stop a while loop, in which case, the else part is ignored.

While loop is often employed if the number of iterations is undefined ab initio.

#Fatimah Oladejobi,
#17/11/2021

```
#Function containing while loop to ask for password
until correct is input.
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

The use of `continue`, `break`, and `pass` keywords	When a **continue** statement is used in a loop, it will stop the current iteration and proceed with the next. Hence, It skips the current code block and returns to the **for** or **while** statement and the control goes back to the start of the loop. When a break statement is used in a loop, it will terminate and exit it before looping through all the items. On the other hand, If the break statement is used in a nested loop, the current loop will be terminated, and the flow will continue with the next line of code. When pass is used in a loop, it is often as a placeholder inside such loops, classes, functions, and if-statements that are meant to be implemented later. Python pass is a **null** statement. When execution begins and the interpreter comes across the pass statement, it does nothing. #Note that; Loops cannot be empty, but if we want a for loop with no content (e.g., has a comment, or we plan to write the code in the future), the pass statement is used to avoid returning an error.

```
#Fatimah Oladejobi,
#17/11/2021
#Examples of Break Statement
#Example 1
#!/usr/bin/python

for letter in 'Fatimah':     # First Example
   if letter == 'm':
      break
   print("Current Letter :", letter)

#Examples of Break Statement
#Example 2
var = 20
while var > 0:
   print("Current variable value :", var)
   var = var -2
   if var == 3:
      break

print("Good bye!")


#Examples of Continue Statement
#Example 1
#!/usr/bin/python

for letter in 'Fatimah':
   if letter == 'm':
      continue

   print("Current Letter :", letter)

#Examples of Continue Statement
#Example 2
var = 20
while var > 0:
   print("Current variable value :", var)
   var = var -2
   if var == 3:
      continue

print("Good bye!")

#Examples of Pass Statement
#Example 1
#!/usr/bin/python
```

```
for letter in 'Fatimah':
   if letter == 'm':
      pass
   print("Current Letter :", letter)

#Examples of Pass Statement
#Example 2
var = 20
while var > 0:
   print("Current variable value :", var)
   var = var -2
   if var == 3:
      pass

print("Good bye!")
```

Infinite loops	Write your description here. Infinite loops refer to code execution that repeats continuously and indefinitely as it lacks functional exit. This is because the condition never becomes False (apparently). Infinite loops are not necessarily bad as observed in client-server programming where the server needs to work with continuity so that the client programs may communicate with the server program whenever the necessity arises. Infinite loops can also come in handy if a new connection needs to be created.

```
#Fatimah Oladejobi,
#17/11/2021
#Example of using break and an infinite loop that can be terminated by user.
# Dice rolling.

import random     #built-in module for random values.
while True:
input("Press enter to roll the dice.") # This line and input not required.
num = random.randint(1, 6) # get a number between 1 to 6
print("Rolled: ", num)
option = input("Roll again? (y/n) ")
# Option for user to break or repeat loop.
if option == 'n':
break # Loop is broken, num is the last rolled value.

#Example of an infinite loop that cannot be terminated by user.
i=1
while(i<11):
    print(i,end='')
```