CSX3001/ITX3001
FUNDAMENTALS OF COMPUTER PROGRAMMING

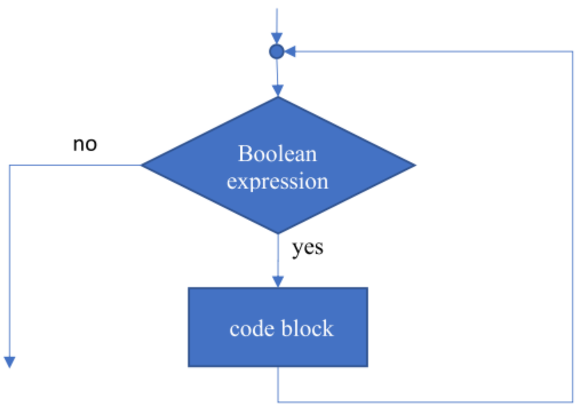
CLASS 04 REPETITION CONTROL STRUCTURE

LOOP CONTROL STRUCTURE, WHILE LOOP, DEFINITE VS. INDEFINITE LOOP,
USING WHILE AND WHILE...ELSE STATEMENTS

PYTHON

REPETITION CONTROL STRUCTURE

Repetition (aka. iteration or looping) control structure is a kind of control that will break the default control structure (sequential control structure) by repeating execution the same block of code (a statement or many statements) several times.



WHILE STATEMENT

```
while Boolean expression:
    code block
```

A ***while*** statement starts with the keyword ***while*** followed by a *condition*, which is always a **Boolean expression**. The computer checks the condition and executes the code block inside the body of ***while*** statement if the condition is *true* and loop back to check the condition again or exits from the loop if the condition is *false*.

In another words, a while statement will keep executing the code block if the condition is evaluated to be *true* otherwise stop and continue with the other statement(s).

What happen if the condition is always evaluated to be true?

Let's try with the follow fragment of code.

```python
count = 0

while count<10 :
    print("Keep printing")

```

What you have just seen when you run the above code is called infinite loop.

How to avoid the infinite loop?

………

………

How to modify the previous code and print out the two words 10 times?

DEFINITE VS. INDEFINTE

In programming, there are two types of iteration which are definite and indefinite loop.

DEFINITE LOOP

Definite loop is a loop that the number of rounds the statement(s) in loop body executed is defined or specified. The number of times the loop body will repeat is known in advance. The previous program, printing the string 10 times is an example of definite loop.

Another Example:

The following fragment of code will print out the first five positive even number.

```python
evenNumber = 0
nTime = 5

while nTime > 0 :
    print(evenNumber)
    evenNumber += 2
    nTime -= 1
```

INDEFINITE LOOP

Indefinite loop is a loop that the number of rounds the loop body will be executed is not explicitly specify. The number of times the loop body will repeat is unknown.

An example of indefinite loop

```
tester = int(input("Enter a positive integer (to terminate
enter a negative integer) : "))

 while tester >= 0:
    print(f"You enter number# {tester}.")
    tester = int(input("Enter a positive integer (to
    terminate enter a negative integer) : "))

print("Exit from a loop.")
```

The above example will keep printing the entered positive number until a negative number is entered.

WHILE…ELSE STATEMENT

```
while Boolean expression :
   code block
else :
   code block
```

In Python, there is an option for ***while*** statement to put ***else*** after the body of ***while*** statement. The code block after this ***else*** will be executed whenever the condition of the ***while*** loop is evaluated to be ***false***.

An Example of while…else statement

```
tester = int(input("Enter a positive integer(to terminate  enter
a negative integer) : "))

 while tester >= 0:
    print("You enter number " + str(tester))
    tester = int(input("Enter a positive integer(to\
                terminate  enter a negative integer) : "))
else:
    print("Exit from a loop")
```

Note: You may notice that there is no difference between previous two examples, therefore when should we use while…else? Consider the following codes:

```
fruits = ["mango", "papaya", "grapes", "banana"]
is_grapes = False
i=0
# iterating through the fruit list
while i< len(fruits):
   if fruits[i] == "grapes":
       is_grapes = True
       print("Grapes Found in the fruits list!")
       break
   i+=1

if is_grapes == False:
   print("No Grapes found in the list!")
```

In the above program, the is_grapes variable is a flag variable. After the loop is executed and if that flag variable is False, then it means there are no “grapes” in the fruits list. If the fruit is found, the flag variable becomes True and we stop the loop and break out of it. With a use of flag variable in the traditional approach, a relevant message is printed.

Now let’s see when we can use while..else. In this scenario, we are going to search for an item without a flag variable which is used in the traditional approach. In the below code, we have iterated through the fruit list. We have used an if condition to check if the current iterator is mango or not. In this code, we have performed searching by utilizing the while..else concept. So we have run the while loop on the condition that the control variable is less than the size. We are then checking whether "mango" is present in the list. If it’s present, we are printing “mango found!” else if it’s not present then the relevant message is printed.

```
fruites = ["papaya","banana","pineapple","mango","grapes"]
i=0
# iterating through the fruit list
while i< len(fruits):
   if fruits[i] == 'mango':
       print("mango found!")
       break
   i+=1
else:
   print("mango not found!")
```

PSEUDOCODE

Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It can be used to explain the sequence of a program as same as flowchart.

An example of using pseudocode,
Let’s start from a programming problem:

Write a program to read in a positive integer number and converse and print out the representation of this number in base 2 (binary number)

This programming problem can be solved by the sequence which is represented by the following pseudocode:

```
Get a positive integer as an input
Declare an empty string for an output
During input is greater than zero keep doing the following:
    -  find the remainder of the input divided by 2
    -  convert the remainder to string and concatenate with
       output string
    -  update input by dividing it with 2 and take only the
       integer part
print out the output
```

which can be directly convert into Python program as follow:

```python
myInput = int(input("Enter a positive integer: "))
binOutput = ""
while myInput > 0:
    binDigit = myInput % 2
    binOutput = str(binDigit) + binOutput
    myInput = int(myInput / 2)
print(binOutput)
```

Can you draw a flowchart to represent the same sequence as this pseudocode?

◆ LOOP EXERCISES

1. Write a Python code that asks a user to enter positive integers until a negative integer is entered and prints a total number of positive integers entered and the sum of all positive integers.

Sample Run#1

```
Enter a positive integer (to terminate enter a negative number): 3
Enter a positive integer (to terminate enter a negative number): 5
Enter a positive integer (to terminate enter a negative number): 10
Enter a positive integer (to terminate enter a negative number): -4
The total number of positive integers entered is 3.
The sum of all positive integer(s) is 18.
```

Sample Run#2

```
Enter a positive integer (to terminate enter a negative number): -1
The sum of all positive integer(s) is 0
```

---

2. Write a Python code that asks a user to enter a value for *n* and lets a user to enter either positive or negative integers *n* times. The code prints the average value of these *n* integers, the sum of all positive integers and the sum of all negative integers. Observe the sample run, you need to increase a number# value for each input.

Sample Run#1

```
How many numbers?  4
Number#1: 12.5
Number#2: -12
Number#3: 13.5
Number#4: -13
The average of these number is 0.25
The sum of all positive integers is 26
The sum of all negative integers is -25
```

Sample Run#2

```
How many numbers?  2
Number#1: 11
Number#2: 19
The average of these number is 15.0
The sum of all positive integers is 30
The sum of all negative integers is 0
```

3. Write a Python code that asks for a value *n* and calculates a sum of all positive integers starting from 1 to *n*. Note that the value of *n* must be greater than zero. Your input/output formatting must strictly follow the below sample runs.

Sample Run#1

```
Enter n: 4
1 + 2 + 3 + 4 = 10
```

Sample Run#2

```
Enter n: 0
The entered value must be greater than ZERO!
```

Sample Run#3

```
Enter n: 7
1 + 2 + 3 + 4 + 5 + 6 + 7 = 28
```

---

4. Write a Python code to compute *n* factorial.

Sample Run#1

```
Enter n: 0
0! = 1
```

Sample Run#2

```
Enter n: 1
1! = 1
```

Sample Run#3

```
Enter n: 4
4! = 1 x 2 x 3 x 4 = 24
```

---

5. Write a Python that takes two integers and store them in the two variables, namely *firstInt* and *secondInt*, and the code prints all integer numbers starting from *firstInt* to *secondInt* on the same line. *Note: If the first integer is less than the second integer, list all numbers in ascending(increasing) order, otherwise in descending (decreasing) order.*

Sample Run#1

```
Enter the first integer: -3
Enter the second integer: 5
Output: -3  -2  -1  0  1  2  3  4  5
```

Sample Run#2

```
Enter the first integer: 3
Enter the second integer: -2
Output: 3  2  1  0 -1 -2
```

---

6. Write a Python code that takes 3 integer numbers, namely *numX*, *numY*, and *numZ*, where *numX* ≥ *numY* ≥ *numZ*, and counts how many integers in a range of *numX* and *numZ* that is divisible by *numY*.

Sample Run#1

```
Enter 3 integer numbers, where as numX ≥ numY ≥ numZ.
numX: 2
numY: 3
numZ: 12
There are 4 numbers in 2...12 that are divisible by 3.
```

Sample Run#2

```
Enter 3 integer numbers, where as numX ≥ numY ≥ numZ.
numX: 10
numY: 5
numZ: 20
There are 3 numbers in 10...20 that are divisible by 5.
```

7. Write a Python code that takes a positive integer and prints out the sum of even digits.

Sample Run#1

```
Input number: 1243
The sum of even digit(s) in 1243 is 6.
```

Sample Run#2

```
Input number: 59144
The sum of even digit(s) in 59124 is 8.
```

---

8. Write a Python code to take a value of *n* and print out *n* stars ("*") on a line.

Sample Run#1

```
n: 2
**
```

Sample Run#2

```
n: 5
*****
```

ASSIGNMENTS

Complete the following exercises in Python IDLE. You must name the python file as

yourid_name_section_class04_as#.py

for example, for assignment 1 will be named,

6310001_Harry_541_class04_as1.py

1. Write a Python code that asks a user to enter a value for *n*, and the code takes *n* floating-point numbers and prints out the first highest and the second highest numbers.

Sample Run#1

```
Enter number of real numbers: 5
Number#1: 45.23
Number#2: 23.5
Number#3: 12
Number#4: 12.321
Number#5: 105.5

The first highest number is 105.5
The second highest number is 45.23
```

Sample Run#2

```
Enter number of real numbers: 1
Number#1: 100.275

The first highest number is 105.5
There is no second highest number.
```

2. Write a Python code that reads in an integer number and prints out an integer number with all digits are placed in reverse order of the input.

Sample Run#1

```
Input number: 1214
Output number: 4121
```

Sample Run#2

```
Input number: 122259
Output number: 952221
```

Note: Python's String and List features are not allowed in solving this problem.

---

3. Write a Python code to take integer number(s) (either positive or negative) until 0 is entered to stop. The code prints the sum odd integers and even integers separately. Furthermore, if the sum of all odd numbers is greater than the sum of even numbers, the code prints "The winner is the sum of all odd numbers." If the sum of all odd numbers is less than the sum of even numbers, the code prints "The winner is the sum of all even numbers." Otherwise, the code prints "No winner here. Try again."

Sample Run#1

```
Number#1: 45
Number#2: 23
Number#3: 12
Number#4: 10
Number#5: 10
Number#6: 0
The sum of all odd numbers is: 68
The sum of all even numbers is: 32
The winner is the sum of all odd numbers.
```

Sample Run#2
Number#1: 5
Number#2: 3
Number#3: 2
Number#4: 4
Number#5: 2
Number#6: 0
The sum of all odd numbers is: 8
The sum of all even numbers is: 8
No winner here. Try again.

4. Write a Python code that takes two integer inputs, namely *numA* and *numB*. Note that *numA* could be higher or lower than *numB*. The code will print integer numbers ranging from a lower integer value to a higher integer value and shall not print integer number(s) that is(are) divisible by 3 and 7. However, if the sum of the printed integer numbers is greater than 5 times of the highest value, the code immediately stops printing the number. For example, if you enter 31 and 14 for *numA* and *numB*, respectively, the output should be

```
14 5 16 17 18 19 20 22
```

Note that 21 is not printed due to the fact that it is divisible by 3 and 7, and the sum value of 14….22 (which is 162) is greater than five times of the highest value (5 * 31 = 155) so the code stops at 22.