

Repetition Structure I

while loop and flow control

Programming I (PRG1)

Diploma in Information Technology

Diploma in Financial Informatics

Diploma in Cybersecurity & Digital Forensics

Common ICT Programme

Year 1 (2019/20), Semester 1

Objectives

At the end of this lecture, you will be able to....

- apply the use of repetition structure in the creation of programs
- code using `while` loop
- alter the program flow of loops with `break` and `continue` statements

Topics

- ❑ What is Repetition Structure
- ❑ while loop
- ❑ Control Flow Statements
 - ❑ break
 - ❑ continue

Repetition Structure

What is a Repetition Structure?

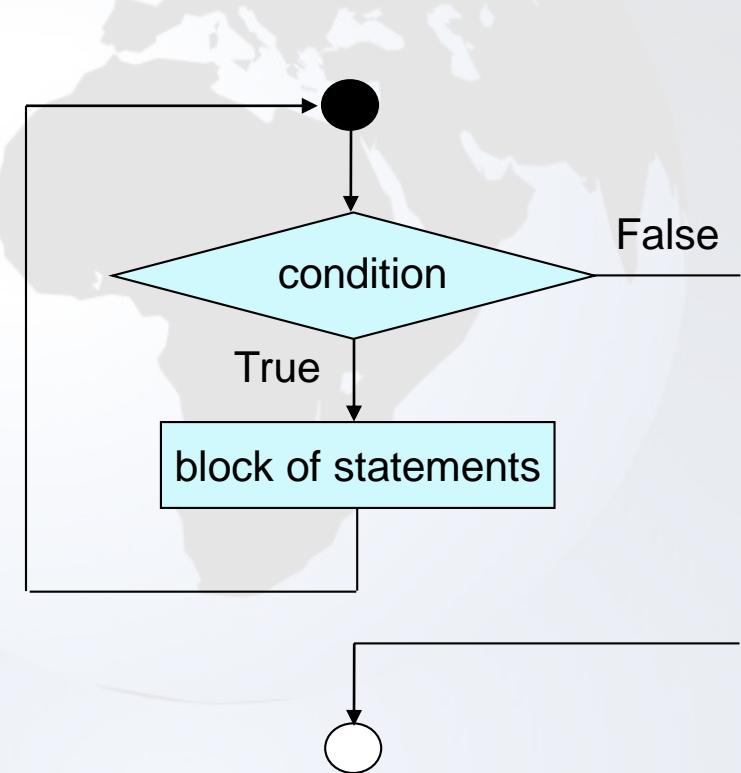
- Execute a block of program statements repeatedly
- 2 types of repetition structure
 - ✓ while loop and for loop
- Examples of repetitive activities:
 - ✓ Repeated prompting of password after incorrect entry
 - ✓ Siri listening for wake-up command
 - ✓ Fitness watch measuring heart rate throughout the day

while loop

while Loop

1. Checks the condition
2. if the condition is *true*, executes the block of statements in the loop
3. Repeat from step 1 until condition is *false*

```
while condition:  
    statement  
    statement
```

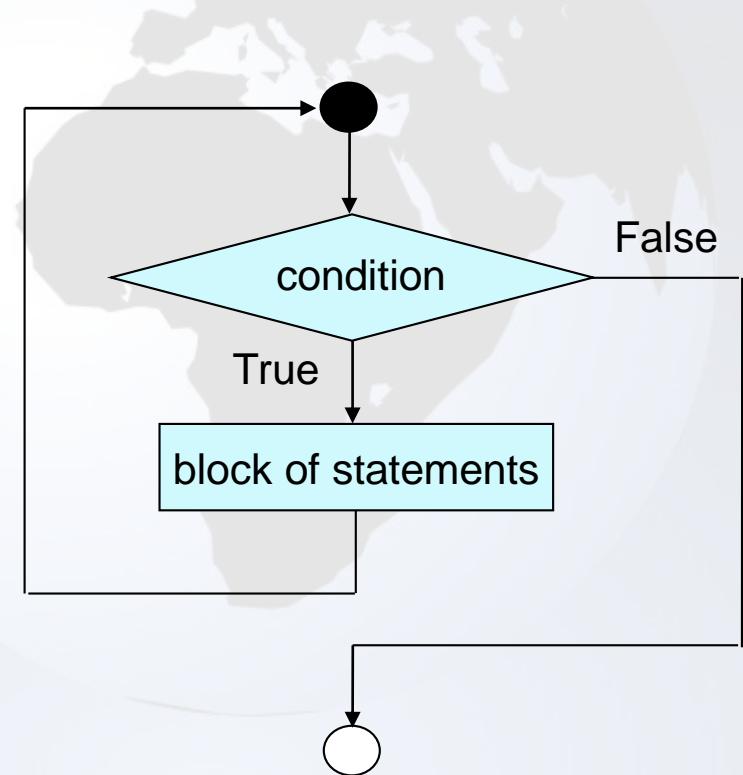


while Loop

- Typical condition statement includes

- ✓ Controlling the *number of iterations* the loop will execute
- ✓ Terminating the loop with *special sequence of character(s)*, also known as **sentinel**

```
while condition:  
    statement  
    statement
```



Example: Control Number of Iterations

Print number 1 to 5

Pseudocode:

1
2
3
4
5

```
SET count to 1
WHILE count less than or equal to 5 THEN
    display count
    increment count
ENDWHILE
```

Python code:

```
count = 1
while count <= 5:
    print(count)
    count = count + 1
```

Terminate with Sentinel

- A **Sentinel value** is also referred to as a flag value, trip value, rogue value, signal value, or dummy data
- It is a special value used as a *condition of termination*, typically in a loop.

Example: Terminate with Sentinel

Prompt the user for pin number until correct pin has been entered

```
pin = 0  
while pin != '12345':  
    pin = input('Enter pin: ')  
print('Correct pin entered')
```

```
Enter pin: 100  
Enter pin: 200  
Enter pin: 12345  
Correct pin entered
```

Example: Terminate with Sentinel

Prompt the user for pin number until correct pin has been entered

```
correctPin = '12345'  
correctEntry = False  
while not(correctEntry):  
    pin = input('Enter pin: ')  
    if(pin == correctPin):  
        correctEntry = True  
    else:  
        print('Incorrect pin. Please try again. ')  
print('Correct pin entered')
```

```
Enter pin: 1  
Incorrect pin. Please try again.  
Enter pin: 5  
Incorrect pin. Please try again.  
Enter pin: 2  
Incorrect pin. Please try again.  
Enter pin: 12345  
Correct pin entered
```

Activity 1: TemperatureSensor.py

A room is installed with a sensor that measures the room temperature at an hourly interval. The temperatures are stored in a list named `temp_list`. Calculate the average temperature reading for the day.

Pseudocode:

```
read measurement #1  
add to total  
  
....  
read measurement #24  
add to total  
average = total / 24
```

Repeated

Activity 1 – TemperatureSensor.py

Previous Program:

```
temp_list = [20.5, 22, 21, 29.3, 28.2, 25, \
             26, 28, 26.3, 25.6, 29.3, 28.4, \
             24.5, 26.3, 25.5, 26.5, 23.3, 24.3, \
             25.4, 26.5, 23.3, 25.4, 26.3, 25.5]

total = 0
total = total + temp_list[0]
total = total + temp_list[1]
total = total + temp_list[2]
...
total = total + temp_list[23]

average = total / 24

print('The average temperature for the day is: \
{:.2f} degree celsius.'.format(average))
```

Repeated code

Activity 1 – TemperatureSensor.py

Modify the program in Activity 1 with the use of the **while** loop

Activity 1 – TemperatureSensor.py

Using Loop:

```
temp_list = [20.5, 22, 21, 29.3, 28.2, 25, \
             26, 28, 26.3, 25.6, 29.3, 28.4, \
             24.5, 26.3, 25.5, 26.5, 23.3, 24.3, \
             25.4, 26.5, 23.3, 25.4, 26.3, 25.5]

total = 0
i = 0
while(i < 24):
    total = total + temp_list[i]
    i += 1

average = total / 24

print('The average temperature for the day is: \
{:.2f} degree celsius.'.format(average))
```

Activity 1 - Trace Table

Condition $i < 24$	Iteration	Value of i after $i += 1$
-	Before start	0
$0 < 24$ True	1	1
$1 < 24$ True	2	2
$2 < 24$ True	3	3
$3 < 24$ True	4	4
:	:	:
$23 < 24$ True	24	24
$24 < 24$ False	Stop	24

Activity 1 – TemperatureSensor.py

Further Improvement:

```
temp_list = [20.5, 22, 21, 29.3, 28.2, 25, \
             26, 28, 26.3, 25.6, 29.3, 28.4, \
             24.5, 26.3, 25.5, 26.5, 23.3, 24.3, \
             25.4, 26.5, 23.3, 25.4, 26.3, 25.5]

total = 0
i = 0
while(i < len(temp_list)):
    total = total + temp_list[i]
    i += 1

average = total / len(temp_list)

print('The average temperature for the day is: \
{:.2f} degree celsius.'.format(average))
```

Activity 2: NumberTable.py

Write a program that prompts the user to enter a number, and print out the math timetable with the use of a `while` loop.

```
Please enter a number: 5
```

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

Infinite Loop

- A loop that will not terminate is called an **infinite loop**.
- Infinite loop occurs as its **condition always evaluates to True**
- Execution of the program can only be stopped by “killing” the program.
- So it is important to make sure the condition of the loop will eventually become false so that the loop will terminate.

Infinite Loop - Examples

```
while True:  
    print("This program will never end!")
```

```
count = 1  
while count <= 10:  
    print("This program will never end!")  
    count = count - 1
```

Can you explain what causes infinite loop in each of these cases?

Activity 3 – Which are Infinite Loops?

Determine if the following code will result in infinite loop?

```
while not(True): print()
```

```
while 0: print()
```

```
count='0'  
while count!=0: print()
```

```
count=0  
while count>=0: print()
```

Control Flow

- **Break**
- **continue**

break Statement

- A **break** statement can be used to *exit from a loop*
- When a **break** statement is executed within a loop,
 - ✓ the loop is terminated
 - ✓ program control is passed to the first statement after the loop.
- A **break** statement is often written in an **if** block when it is used in a loop

break Statement - Example

```
while True:  
    word = input('Please enter a word (anything else to quit): ')  
    if word.isalpha():  
        print ('The word was ' + word)  
    else:  
        print ('Exiting...')  
        break
```

```
Please enter a word (anything else to quit): I  
The word was I  
Please enter a word (anything else to quit): Love  
The word was Love  
Please enter a word (anything else to quit): Programming  
The word was Programming  
Please enter a word (anything else to quit): !  
Exiting...
```

continue Statement

- A **continue** statement can be used to *skip to the next iteration of the loop*
- When a **continue** statement is executed within a loop,
 - ✓ the current iteration is terminated. Rest of code in current iteration is skipped.
 - ✓ program control continues on to next iteration if condition is true.
- A **continue** statement is often written in an **if** block when it is used in a loop

continue Statement - Example

```
while True:  
    word = input('Please enter a word: ')  
    if word.isalpha():  
        print('The word was ' + word)  
    else:  
        print ('You did not enter a word, Please try again.')  
        continue  
    print('Loop again!')
```

```
Please enter a word: I  
The word was I  
Loop again!  
Please enter a word: miss  
The word was miss  
Loop again!  
Please enter a word: Hotel  
The word was Hotel  
Loop again!  
Please enter a word: Jen  
The word was Jen  
Loop again!  
Please enter a word: !  
You did not enter a word, Please try again.  
Please enter a word:
```

Activity 4 – NumberGuessing.py

- Write a program that simulates a number guessing game. It first generates a random number between 1 and 100. It then prompts user to guess the correct number. User can enter -1 to end the game or the game will end after 5 tries.
- Sample outputs of the program:

```
Welcome to Number Guessing Game
Try 1: Enter a number between 1 and 100 (or -1 to end): 45
45 is too low.
Try 2: Guess again, enter a number between 1 and 100 (or -1 to end): 79
79 is too high.
Try 3: Guess again, enter a number between 1 and 100 (or -1 to end): 60
60 is too low.
Try 4: Guess again, enter a number between 1 and 100 (or -1 to end): 72
Bingo, you've got it right!

Bye-bye!
```

Activity 4 – NumberGuessing.py

```
Welcome to Number Guessing Game
```

```
Try 1: Enter a number between 1 and 100 (or -1 to end): 80  
80 is too high.
```

```
Try 2: Guess again, enter a number between 1 and 100 (or -1 to end): 20  
20 is too low.
```

```
Try 3: Guess again, enter a number between 1 and 100 (or -1 to end): 60  
60 is too high.
```

```
Try 4: Guess again, enter a number between 1 and 100 (or -1 to end): 40  
40 is too high.
```

```
Try 5: Guess again, enter a number between 1 and 100 (or -1 to end): 45  
45 is too high.
```

```
Game over. The correct answer is: 34
```

```
Bye-bye!
```

Summary

- ❑ Repetition structure allows statements to be repeated until certain condition failed
- ❑ The while loop checks the condition first to decide whether to enter the loop
- ❑ The break statement breaks out of the loop completely.
- ❑ The continue statement skips the remaining part of the current iteration and continues to the next iteration.

Reading Reference

❑ Python 3.6.x Documentation

❑ https://docs.python.org/3.6/reference/compound_stmts.html#while

❑ Learn Python Tutorial

❑ <https://www.learnpython.org/en/Loops>