MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Computer Science Department

COMP8060 – Scientific Programming with Python

Week 5 – Iteration structures

Dr. Bruno Andrade

October, 2025

www.mtu.ie

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

- **Summary:**
  - Introduction to Repetition Structures
  - The while Loop: a Condition-Controlled Loop
  - The for Loop: A Count-Controlled Loop
  - Nested Loops
  - List and Dict comprehension
  - Groups!

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

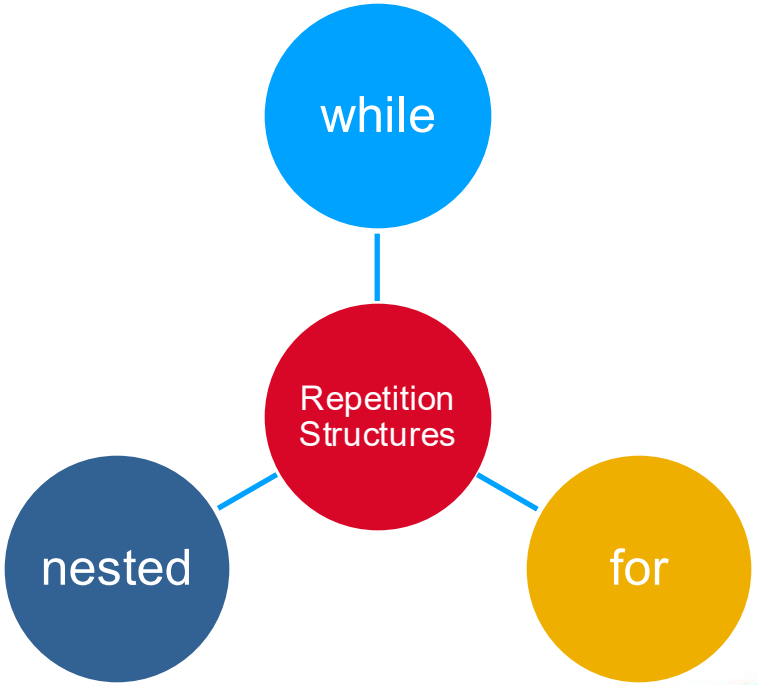
Introduction to Repetition Structures

Introduction to Repetition Structures

- Sometimes we need a code that performs the same task multiple times
  - Disadvantages to duplicating code
    - Make programs large
    - Time consuming
    - May need to be corrected in many places
    - It makes your code complicated
  - All programming languages provide a construct that support iterative actions

Introduction to Repetition Structures

- These constructs are collectively known as Repetition Structures.

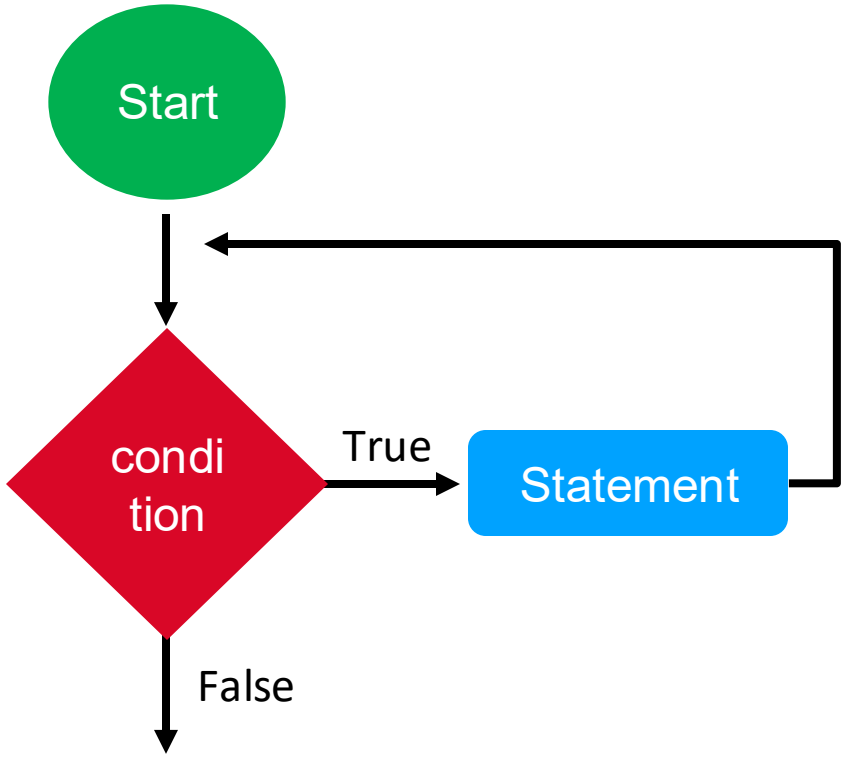


The while loop

The `while` Loop: a Condition-Controlled Loop

- `while` loop: while condition is true, do something
  - Two parts:
    - **Condition** tested for true or false value
    - **Statements** repeated if the condition is true

```
while condition:
    statements
```



The `while` Loop: a Condition-Controlled Loop

- Iteration: one execution of the body of a loop

```
num = 5

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

```
5
6
7
8
9
```

- `while` loop is known as a *pretest* loop
  - Tests condition before performing an iteration
    - Will never execute if condition is false to start with
    - Requires performing some steps prior to the loop

The `while` Loop: a Condition-Controlled Loop

```
num = 5

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

```
6
7
8
9
10
```

- Typically, last statement is increment of counter

The `while` Loop: a Condition-Controlled Loop

```
num = 5

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

- num is the *control variable*
- A control variable must change at some point during the loop otherwise the condition will not change

If the condition is unchangeable, you got yourself in an infinite loop.

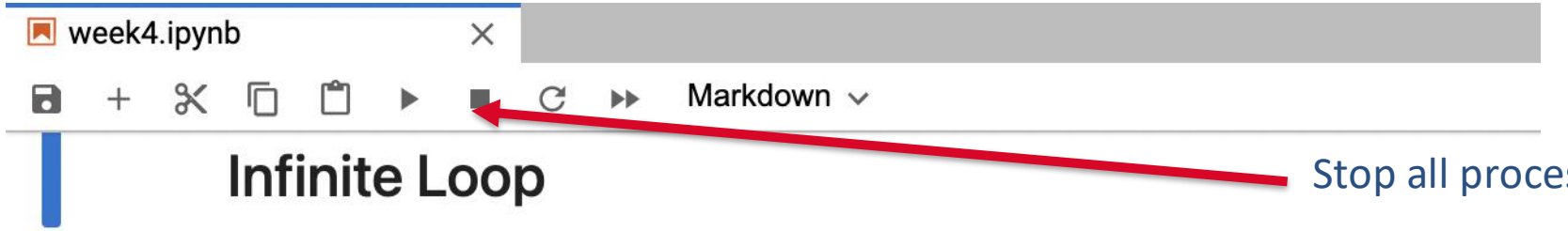
Infinite Loops

- Loops **must** contain within themselves a way to terminate
  - Something inside a `while` loop must eventually make the condition false

- Infinite loop: loop that does not have a way of stopping
  - Repeats until program is interrupted

```
stop = False

while stop == False:

  continueLooping = input( 'Do you want to continue looping y/n')
```

Infinite Loops

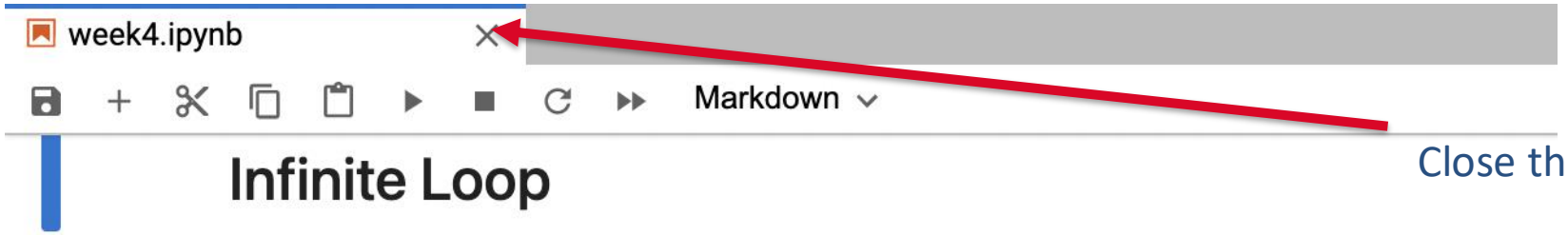



```
[1]: stop = False

while stop == False:

    continueLooping = input( 'Do you want to continue looping? (y/n)')
```

```
Do you want to continue looping? (y/n) y
Do you want to continue looping? (y/n) y
Do you want to continue looping? (y/n) n
```

Terminating an infinite loop

- If nothing works, click in the x to close the notebook.



Infinite Loop

```
[1]: stop = False

while stop == False:

    continueLooping = input( 'Do you want to continue looping? (y/n)')
```

```
Do you want to continue looping? (y/n) y
Do you want to continue looping? (y/n) y
Do you want to continue looping? (y/n) n
```

Fixing an infinite loop

```
stop = False

while stop == False:

    continueLooping = input( 'Do you want to continue looping? (y/n)')
    if continueLooping == 'n':
        stop = True
```

```
Do you want to continue looping? (y/n) n
```

Example Infinite While Loop

```
num = 5

while num < 10:
    print (num)
```

Is the example below an infinite While Loop

```
num1 = 5
num2 = 20

while num1<20 or num2>5:
    print (num1+num2)
    num1 = num1-1
    num2 = num2-1
```

Beware the infinite loop. If a process is taking forever to finish, it was probably caught in it.

Using Break to exit a loop

- To exit a while loop immediately without running any remaining code in the loop, regardless of the results of any conditional test, use the break statement.

```
prompt = "\nPlease enter the name of a city you have visited:"
prompt += "\n(Enter 'quit' when you are finished.) "
while True:
    city = input(prompt)
    if city == 'quit':
        break
    else:
        print(f"I'd love to go to {city.title()}!")
```

The Augmented Assignment Operators

- In many assignment statements, the variable on the left side of the = operator also appears on the right side of the = operator
  - num = num + 5
- Augmented assignment operators: special set of operators designed for this type of job
  - Shorthand operators

Table 5-2 Augmented assignment operators

Operator	Example Usage	Equivalent To
+=	x += 5	x = x + 5
-=	y -= 2	y = y - 2
*=	z *= 10	z = z * 10
/=	a /= b	a = a / b
%=	c %= 3	c = c % 3

Example

```
num = 5

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

```
num = 5

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

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

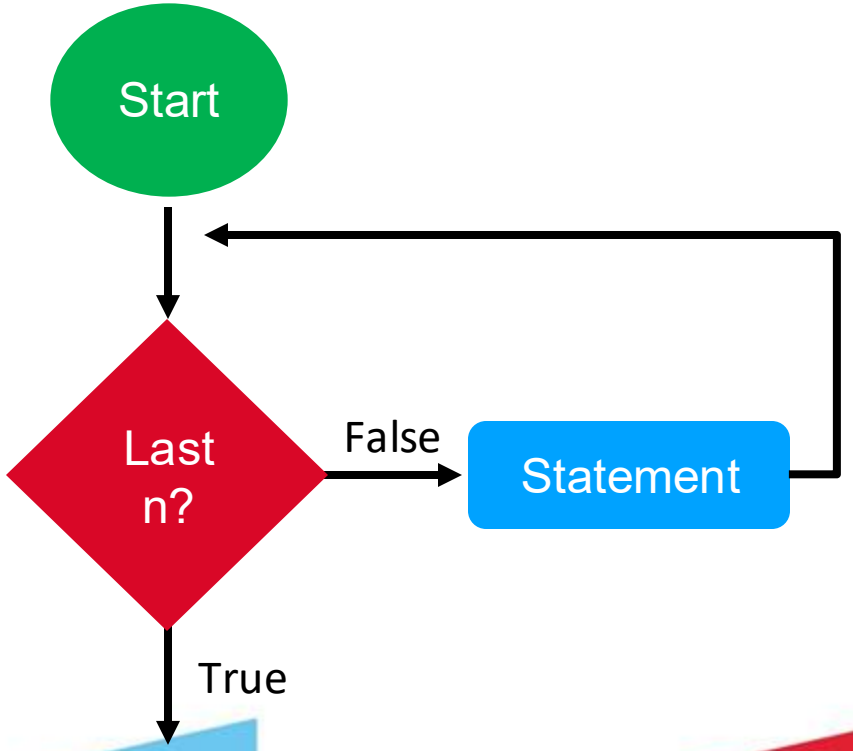
The for loop

The `for` Loop: a Count-Controlled Loop

- Count-Controlled loop: iterates a specific number of times
  - Use a for statement to write a count-controlled loop
    - **Designed to work with sequence of data items**
      - Iterates once for each item in the sequence
    - General format:

for *variable* in *[val1, val2, etc]*:

statements

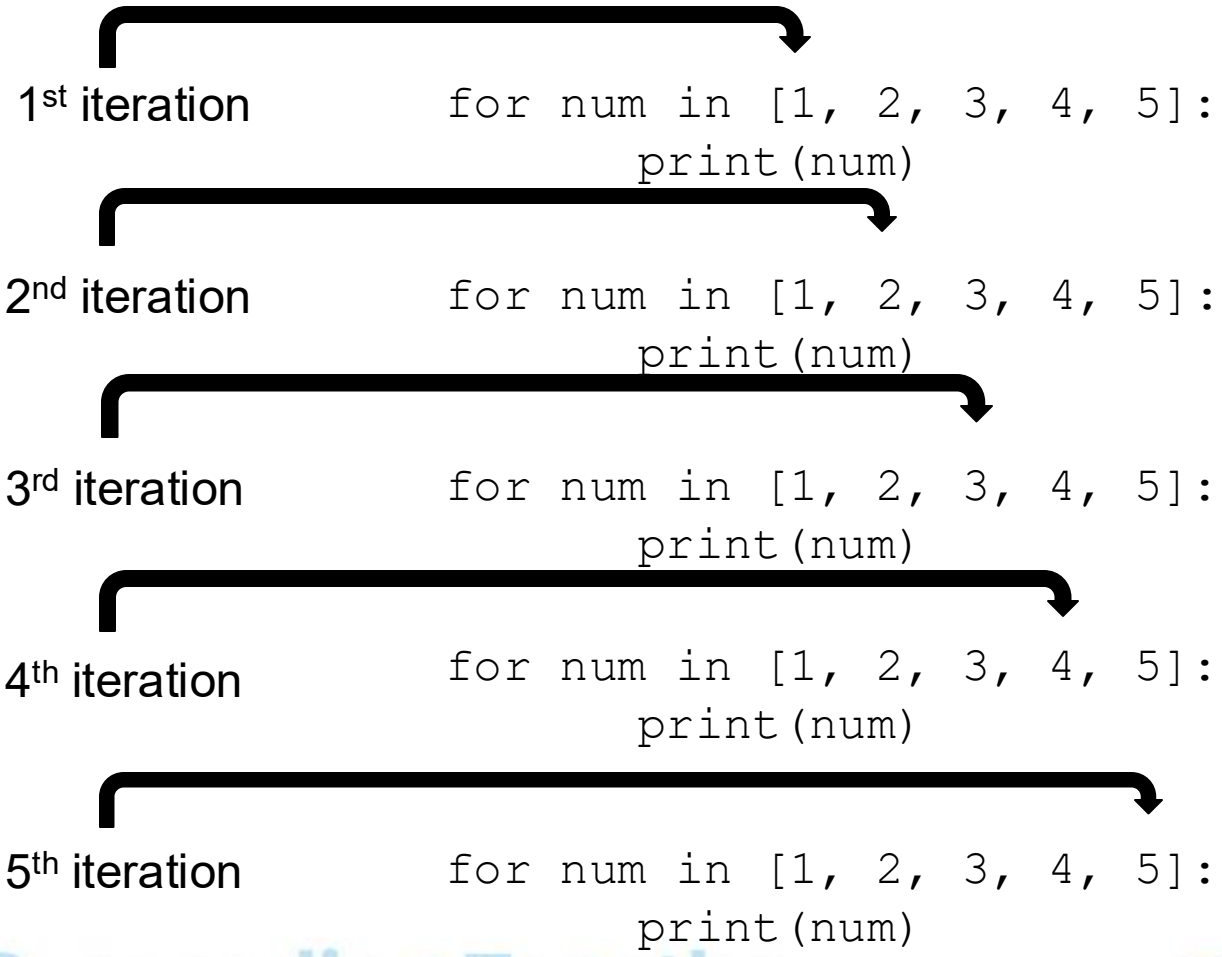


for Loop Example

```
print ('I will display the numbers 1 through 5.')
for num in [1, 2, 3, 4, 5]:
    print (num)
```

```
I will display the numbers 1 through 5.
1
2
3
4
5
```

for Loop Example




```
for num in [1, 2, 3, 4, 5]:
    print (num)
```

```
1
2
3
4
5
```

for Loop Example

```
for word in ["Hello", "There", "Everyone"]:
    print (word)
```

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Using the range function for loops

Using the `range` Function with the `for` Loop

- The `range` function simplifies the process of writing a `for` loop
  - You can think off the range function as generating a list of numbers, which is generally used to iterate over with for loops.

```
for num in range (5) :
    print (num)
```

```
0
1
2
3
4
```

Using the `range` Function with the `for` Loop

```
range([start,] stop [,step])
```

- `range` characteristics:
  - **One argument**: used as **stopping** limit (not inclusive)
  - **Two arguments**: starting value and ending limit (not inclusive of the ending value)
  - **Three arguments**: third argument is step value

Using the `range` Function with the `for` Loop

- `range` characteristics (all arguments **must be** integer):
  - One argument `range(stop)`: used as ending limit

    `0, 1, …, stop-1`
  - The default value for `start` is 0 and the default `step` is 1

Using the `range` Function with the `for` Loop

- In a for loop, the purpose of the target variable is to reference each item in a sequence of items as the loop iterates.

- In many situations it is helpful to use the target variable in a calculation or other task within the body of the loop.

```
for number in range(1, 9):
    square = number**2
    print (number, ' \t ' , square)
```

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

Using the `range` Function with the `for` Loop

- If you pass a third argument to the range function, that argument is used as a step value.

```
for num in range(1, 10, 2):
    print (num)
```

```
1
3
5
7
9
```

Letting the User Control the Loop Iterations

- We may have situations where we want the user to specify the number of iterations of a loop.

- Can **receive range inputs from the user**, place them in variables, and call the range function in the for clause using these variables
  - Be sure to consider the end cases: range does not include the ending limit

User Controlled for Loop - example

```
print ('Program will print out squares of a sequence of values')
end = int(input('How high should I go? '))
start = int(input('Enter the starting number:'))

print ('Number \t Square')
print ('-----------------')


for number in range ( start, end + 1 ) :
    square = number**2
    print (number, ' \t ' , square)
```

```
How high should I go? 4
Enter the starting number:2
Number      Square
----------------
2           4
3           9
4           16
```

Generating a reverse range

- The `range` function can be used to generate a sequence of numbers in descending order
  - How would I output the following sequence of numbers

```
for num in range (10, 0, -1):
    print (num)
```

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

Make sure starting number is larger than end limit, and step value is negative

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Nested Loops

Nested Loops

- Nested loop: A loop inside another loop (remember nested if statements)

- Key points about nested loops:
  - Inner loop goes through all iterations for each iteration of outer loop
  - Example: **analog clock** works like a nested loop
    - Minutes hand moves 60 times for each movement of the hours hand: for each iteration of “hours” do 60 iterations of “seconds”
  - Total number of iterations in nested loop?

Nested Loop Example

- Implement a program that will print out all possible times for the hours, minutes and seconds within a single day

```
for hours in range(24):
    for minutes in range(60):
        for seconds in range(60):
            print (hours, ' : ' ,minutes, ' : ' , seconds)
```

Don’t run this code, it will take forever.

Nested For Loop Example

```
for num1 in range(3):
    for num2 in range(2):
        print (num1+num2)
```

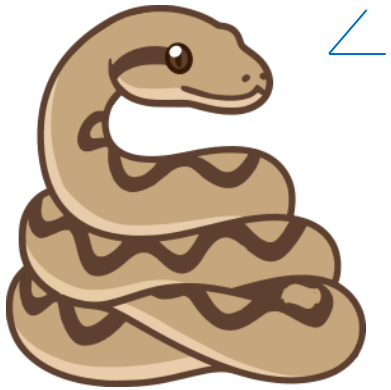
```
0
1
1
2
2
3
```

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

List and Dict Comprehension

An elegant way to iterate.

- Python is an object-oriented programming language.
- It is an interpreted, powerful, and syntactically easy language. Its easy syntax for complicated coding enhances the productivity of programmers, and although there are many ways to build a code, there is always the optimal way, which normally it's the obvious way.
- The optimal way of coding in Python is known as the Pythonic way.



An elegant way to iterate.

- Iterate through an object could be messy and your code will be harder to maintain.
- Python has an easier way to solve this issue using List Comprehension.
- List Comprehension is an elegant way to define, create and iterate through lists. Let's check an example:

```
h_title=[]

for letter in "I Love Python":
    h_title.append(letter)

print(h_title)
```

```
['I', ' ', 'L', 'o', 'v', 'e', ' ', 'P', 'y', 't', 'h', 'o', 'n']
```

An elegant way to iterate.

- Now with List Comprehension:

```
h_title = [letter for letter in "I Love Python"]
print(h_title)
```

```
['I', ' ', 'L', 'o', 'v', 'e', ' ', 'P', 'y', 't', 'h', 'o', 'n']
```

- We reduced the code to a single line, also known as a one-liner.

An elegant way to iterate.

- It looks clumsy, but let's break it down.

```
h_title = [letter for letter in "I Love Python"]
```

NewList = [expression for item in iterable]

List comprehension for filtering data.

- List comprehensions can also be used for filtering.
- For example, if we wanted to select only elements with more than 2 characters in a list and display them in uppercase:

```
strings = ["I","love","Python", "more", "than","C"]
filtered_list = []

for whatever in strings:
    if len(whatever)>2:
        filtered_list.append(whatever.upper())
print(filtered_list)
```

```
['LOVE', 'PYTHON', 'MORE', 'THAN']
```

List comprehension for filtering data.

- We can rewrite it using a list comprehension:

```
strings = ["I","love","Python", "more", "than","C"]

filtered_list = [whatever.upper() for whatever in strings if len(whatever) >2]
print (filtered_list)
```

```
['LOVE', 'PYTHON', 'MORE', 'THAN']
```

List comprehension for filtering data.

```
[whatever.upper() for whatever in strings if len(whatever) >2]
```

NewList = [expression for item in collection if condition is **True**]

Dict comprehension.

- Dict comprehension works in a similar fashion to list comprehension, but changing the syntax a little bit to fit the structure of a dictionary.
- Let's say we want to create a dictionary containing a squared table:

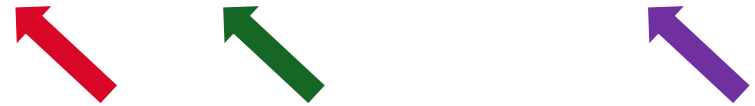
```
squared_dict = {n: n**2 for n in range(1,11)}
squared_dict
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

Dict comprehension.

- Let's break it down:

```
{n: n**2 for n in range(1,11)}
```



NewDict = [key-expr: value-expr for item in collection]

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Groups

Groups - A

Group 1A

Shahmeer Khan
Chetan Babu Mahendiran
David Compagno
Michéal Kennedy

Group 2A

Bramadet Subhash
Afreeda Shireen
Soma Siva Saran Teja Peddireddy
Alvin Caleb Nchoga

Group 3A

Abhinay Revadala
Joan Henry Francis
Aaron W. Walsh
Ajaykumar Borigorla

Group 4A

Maria Chipanera
Sai Saketh Sharma Viriventi
Lynal Ivan Pinto
Yasara Ekanayake

Group 5A

Muhammad Arslan Akram
Arjun Krishna Krishnakumar
Rini Carmel Mary Raja Mohana Lalitha
Krushna Sanjay Jadhav

Groups - B

Group 1B

Suresh Prabu Maithreyan
Odumeh Victory
Saju Joyal
Ehimare Ose B.

Group 2B

Joshi Vaishnavi Raghav
Raju Shivaani
Katla Saikumar
Dharane Rishikesh Mahendra

Group 3B

Magam Mudalige Sharmila P.
Sridhar Monesh
Belvatha Krishnaiah Megha
Mende Jagadeesh

Group 4B

Tariq Ilhan
Gamua Elvis Tachu
Okoli Victor Chibundo
Khan Taha

Group 5B

Vashist Yash
Butler Sophie
Vayigandla Devarsh
Miranda Justin Rithesh

Announcement

- You might be thinking…

Prize

- The winning team of each lab groups will be the first ones selecting a project from the assessment 2 project pool!!.

Exercises

Exercises

1) Write a program that will ask the user to continually guess a value between 1 and 10. When the user guesses the correct value the program should print out a message reporting they have been successful.
2) Write a program for a Coffee shop to sell products such as coffee, tea and scones. Allow the user to select more than one product and print the final price and items selected.
3) Write a program that will allow the user to specify a start and stop numerical value. It should then display each number between the start and stop numerical values (should include both start and stop) and inform the user if it the sum of this range of numbers is an even or odd number.
4) Write an application that asks the user for a starting height of a bouncing ball. At each bounce the ball halves it's height. Display the height and number of each rebound bounce, stopping when the ball is 1mm off the ground.

Things to consider…

- It must stop when it is 1mm or less off the ground.
- 1mm = 0.001m so is our controlling value in the loop
- i.e. while the height is greater than or equal to 0.001 keep iterating the loop

That’s all folks!