Lecture 12 – Loops – while loops
At the end of this lecture, students should:

- understand the concept of a loop for defining repeated tasks
- understand the structure of a while loop, i.e.,
  - the loop initialisation
  - the body of the loop
  - the loop condition
  - the loop increment
- be able to design and write Python while loops
Control structures

- We want to understand how the computer works its way through a program, finding which instruction to execute next.

- Control structures allow us to change the flow of statement execution in our programs. So far we have looked at selection statements (if statements). Selection or if statements are also called branch statements, as, when the program arrives at an if statement, control will "branch" off into one of two or more "directions".

- Now we will look at another control structure, iteration. Iteration means that the same code is executed repeatedly.

- Some examples where iteration is required are:
  - User login – asking for the password until the correct one is given
  - Menu option control – menu options are repeatedly displayed and processed until the ‘exit’ option was selected
- We use loops to implement iteration.
- How does the while loop execute?

\[ \text{while boolean_expression:} \\
\text{statement1} \\
\text{statement2} \\
\text{statement3} \\
\ldots \]

First, the condition is tested.

If the condition is true, the loop statements (the loop body) are executed.

After the loop statements (loop body) have been executed, control returns to top of the loop, and the condition is tested again.

As long as the condition is true, the loop statements will be executed.
while loop - example

```python
def print_lines():
    count = 0
    while count < 100:
        print("Programming is fun!")
        count = count + 1

def main():
    print_lines()

main()
```

Programming is fun!
Programming is fun!
Programming is fun!
...
Programming is fun!

When the condition becomes False the control moves on to the code after the loop.
while loop - terminology

```
def print_lines():
    count = 0
    while count < 100:
        print("Programming is fun!")
        count = count + 1
    print("Done!")

def main():
    print_lines()
main()
```

**Initialisation**: anything which needs to be done before the loop starts.

**body**: the statements which are to be executed over and over (or not at all).

**condition**: a boolean expression which is test repeatedly to determine whether the body of the loop should be executed or not.

**increment**: this changes the loop variable so that eventually the condition becomes false. Remember that a loop will only stop when the condition is false.
Sometimes we don't need an overt increment statement, e.g.,

```python
def total_user_numbers():
    total = 0
    number = int(input("Enter a number (0 to end): "))
    while number != 0:
        total += number
        number = int(input("Enter a number (0 to end): "))
    print("Total: ", total)

def main():
    total_user_numbers()
main()
```

Enter a number (0 to end): 5
Enter a number (0 to end): 6
Enter a number (0 to end): 2
Enter a number (0 to end): 4
Enter a number (0 to end): 0
Total: 17
### Suppressing the new line after printing

- The `print()` function has an optional last argument, `sep = "..."` which can be used to change the separator between the arguments of the `print()` statement.

```
print(1, "Meravigioso", "Fabulous", sep = "*")
print('The final results are:', 56, "and", 44, sep = "")
```

```
1*Meravigioso*Fabulous
The final results are:56and44
```

- The `print()` function has an optional last argument, `end= "..."` which can be used to change the new line character which is inserted after the arguments have been printed.

```
print("The", end= " ")
print("cat", end= "*")
print("said", end= "")
print("nothing", end= "!")
print()
```

```
The cat*saidnothing!
```
Complete the function

- For an integer, a divisor is a number which divides exactly into the integer (a factor of the integer), e.g., the divisors of 6 are 1, 2, 3, 6. Complete the get_all_divisors() function. Note that 1 and the number itself are divisors (as they divide into the number exactly).

```python
def get_divisor_string(number):
    #Add the code here

def main():
    print(get_divisor_string(24))
    print(get_divisor_string(25))
    print(get_divisor_string(5628))
main()
```

1 2 3 4 6 8 12 24
1 5 25
1 2 3 4 6 7 12 14 21 28 42 67 84 134 201 268 402 469 804 938 1407 1876 2814 5628
def get_divisor_string(number):
    divisor = 1
    div_string = ""
    while divisor <= number // 2:
        if number % divisor == 0:
            div_string += str(divisor) + " "
        divisor += 1
    div_string += str(number)
    return div_string

def main():
    print(get_divisor_string(24))
    print(get_divisor_string(25))
    print(get_divisor_string(5628))

main()
A perfect number is an integer that is equal to the sum of its divisors (excluding the number itself), e.g., \(28 = 1 + 2 + 4 + 7 + 14\).

Complete the following two functions. The `check_perfection()` function checks for perfection and prints either ' # is a perfect number' or ' # is NOT a perfect number'.

```python
def get_divisor_sum(number):
    #Add the code here

def check_perfection(number):
    #Add the code here

def main():
    check_perfection(28)
    check_perfection(54)
    check_perfection(496)

main()
```

28 is a perfect number
54 is NOT a perfect number
496 is a perfect number
Complete the function

def get_divisor_sum(number):
    divisor = 1
    div_sum = 0
    while divisor <= number // 2:
        if number % divisor == 0:
            div_sum += divisor
        divisor += 1
    return div_sum

def check_perfection(number):
    if number == get_divisor_sum(number):
        print(number, "is a perfect number")
    else:
        print(number, "is NOT a perfect number")

def main():
    check_perfection(28)
    check_perfection(54)
    check_perfection(496)

main()

28 is a perfect number
54 is NOT a perfect number
496 is a perfect number
In a Python program:

- a loop is used for implementing repetition
- a loop has four parts
  - the loop initialisation
  - the body of the loop
  - the loop condition
  - the loop increment
- a while loop has the following syntax:

  ```python
  while boolean_expression:
      statement1
      statement2
      ...  
  ```
Examples of Python features used in this lecture

```python
def get_divisor_sum(number):
    divisor = 1
    div_sum = 0

    while divisor <= number // 2:
        if number % divisor == 0:
            div_sum += divisor
            divisor += 1

    return div_sum

def fun_stuff():
    count = 0
    while count < 4:
        print("Programming is fun!")
        count = count + 1
```