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
From lecture 11

- the if block of an if...else statement is executed only if the boolean expression evaluates to True, otherwise the else block is executed.
- if...elif statements are useful if there is a situation where at most one option is to be selected from many options. The if...elif statement has an optional final else part.

```python
def get_random_horoscope()
    number = random.randrange(0, 10)
    if number < 4:
        return "Amazing day ahead"
    elif number < 7:
        return "Romance is very likely"
    elif number < 8:
        return "Proceed with caution"
    return "Lucky lucky you"

def main():
    print("Today's message:", get_random_horoscope())
    print("Today's message:", get_random_horoscope())

main()
```
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?

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.
```python
1 def print_lines():
2     count = 0
3     while count < 100:
4         print("Programming is fun!")
5         count = count + 1
6 def main():
7     print_lines()
8 main()
```

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

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>def print_lines():</td>
<td>initialisation</td>
</tr>
<tr>
<td>2</td>
<td>count = 0</td>
<td>condition</td>
</tr>
<tr>
<td>3</td>
<td>while count &lt; 100:</td>
<td>body</td>
</tr>
<tr>
<td>4</td>
<td>print(&quot;Programming is fun!&quot;)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>count = count + 1</td>
<td>increment</td>
</tr>
<tr>
<td>6</td>
<td>print(&quot;Done!&quot;)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>def main():</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>print_lines()</td>
<td>Programming is fun!</td>
</tr>
<tr>
<td>9</td>
<td>main()</td>
<td>Programming is fun!</td>
</tr>
</tbody>
</table>

**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
def show_output():
    number = 1
    count = 10
    value = 4

    while count > 4:
        count = count - 2
        print(str(number) + ":", count, value)
        value += count
        number += 1

    print()
    print(str(number) + ":", count, value)

def main():
    show_output()

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

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

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

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):

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
Complete the function

- The `get_dice_throws_result()` function throws a number of dice (given by `num_dice_throws`) and counts how often the dice value, `dice_to_check` occurs. Complete the function.

```python
def get_dice_throws_result(num_dice_throws, dice_to_check):

def main():
    print(get_dice_throws_result(30000, 6), "sixes thrown (out of 30000 throws")
    print(get_dice_throws_result(6, 6), "sixes thrown (out of 6 throws")
    print(get_dice_throws_result(600000, 6), "sixes thrown (out of 600000 throws")

main()
```

4913 sixes thrown (out of 30000 throws)
0 sixes thrown (out of 6 throws)
99929 sixes thrown (out of 600000 throws)
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):

def check_perfection(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
Complete the function

- The get_legal_number() function repeatedly prompts the user for a number until the user number is within (both inclusive) the two numbers passed as parameters. The function returns the user number. Complete the function:

```python
def get_legal_user_num(lower, upper):
    prompt = "Enter a number (" + str(lower) + "-" + str(upper) + "): "

def main():
    print(get_legal_user_num(0, 6))
    print(get_legal_user_num(10, 20))
    print(get_legal_user_num(1, 2))

main()
```
Complete the function

- The following function keeps prompting the user to guess a hidden number until the user correctly guesses the number. At each guess the function lets the user know if the guess is too high or too low. The function also keeps track of (and prints) the number of guesses. Complete the user_number_guess() function:

```python
def user_number_guess(computer_num):
    prompt = "Enter your guess (1 - 99): "

    user_number_guess(random.randrange(1, 100))

main()
```

```python
Enter your guess (1 - 99): 50
Too high
Enter your guess (1 - 99): 25
Too high
Enter your guess (1 - 99): 13
Too low
Enter your guess (1 - 99): 20
Too low
Enter your guess (1 - 99): 23
Correct! Number of guesses: 5
```
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
```