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

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

It is important to understand how the computer works its way through the program statements, i.e., the order in which instructions are executed.

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
Iteration – while loops

We use loops to implement iteration. How does the while loop execute?

```
while boolean_expression:
    statement1
    statement2
    statement3
    ...
```

First, the condition is tested.
If the condition evaluates to True, the loop statements (the loop body) are executed.
After the loop statements have been executed, control returns to top of the loop, and the condition is tested again.
As long as the condition evaluates to True, the loop statements are executed.
while loop - example

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

def main():
    print_lines()

main()
```

When the condition becomes False the control moves on to the code after the loop (line 6 in the code above).
while loop - terminology

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

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

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

**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.

```python
1 def print_lines():
2     count = 0  # initialisation
3     while count < 100:
4         print("Programming is fun!")  # body
5         count = count + 1  # increment
6     print("Done!")

7 def main():
8     print_lines()
9 main()
```

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

CompSci 101 - Principles of Programming
while loop - terminology

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 = 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 display_output():
    number = 1
    count = 10
    value = 4

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

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

def main():
    display_output()

main()
Complete the function

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

```python
import random
def get_dice_throws_result(num_throws, dice_to_check):
    pass

def main():
    print("30000 throws,", get_dice_throws_result(30000, 6), "sixes")
    print("6 throws,", get_dice_throws_result(6, 6), "sixes")
    print("600000 throws,", get_dice_throws_result(600000, 5), "fives")
main()
```

30000 throws, 4913 sixes
6 throws, 0 sixes
600000 throws, 99929 fives
Suppressing the new line after printing

The print() function has an optional argument, sep = "..." which can be used to set the separator between the arguments of the print() statement (the default is a blank space).

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

```
The final results are:56and44
```

The print() function has an optional argument, end= "..." which can be used to set the character/2 which is/are to be inserted after the arguments have been printed (the default is a new line character).

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

```
The cat*saidnothing!
Enough said!
```
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. Note that 1 and the number itself are divisors (they divide into the number exactly). Complete the string_of_divisors() function.

```python
def string_of_divisors(number):

def main():
    string_of_divisors(24)
    string_of_divisors(25)
    string_of_divisors(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

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. Note that 1 and the number itself are divisors (they divide into the number exactly). For this function we only want the sum of all the divisors less than the number itself. Complete the function.

```python
def get_sum_of_divisors(number):
    divisors = [i for i in range(1, number) if number % i == 0]
    return sum(divisors)

def main():
    print("get_sum_of_divisors(6)", get_sum_of_divisors(6))
    print("get_sum_of_divisors(24)", get_sum_of_divisors(24))
    print("get_sum_of_divisors(25)", get_sum_of_divisors(25))
    print("get_sum_of_divisors(5628)", get_sum_of_divisors(5628))
main()
```
Complete the function

A perfect number is an integer that is equal to the sum of its divisors (including 1, excluding the number itself), e.g., the sum of the divisors of 28 is 28 (1 + 2 + 4 + 7 + 14). Complete the `check_perfection()` function which checks for perfection and prints either '# is a perfect number' or '# is NOT a perfect number'.

```python
def get_sum_of_divisors(number):
    #See the previous Slide

def check_perfection(number):
    message_is = "is a perfect number"
    message_is_not = "is NOT a perfect number"

def main():
    check_perfection(28)  # 28 is a perfect number
    check_perfection(54)  # 54 is NOT a perfect number
    check_perfection(496) # 496 is a perfect number

main()
```
Complete the function

Complete the user_number_guess() function which 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. At the end, the function also prints the number of guesses taken.

```python
def user_number_guess(computer_num):
    prompt = "Enter your guess (1 - 99): ",
    num_guesses = 0

    while num_guesses != 0:
        user_guess = int(input(prompt))
        if user_guess > computer_num:
            print("Too high")
            num_guesses += 1
        elif user_guess < computer_num:
            print("Too low")
            num_guesses += 1
        else:
            print("Correct! Number of guesses: ", num_guesses)
            break

def main():
    user_number_guess(random.randrange(1, 100))

main()
```
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) + ")
    prompt = prompt + "): ": 

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

main()
```

Enter a number (0-6): -1
Enter a number (0-6): 8
Enter a number (0-6): 6
Enter a number (10-20): 13
Enter a number (1-2): 3
Enter a number (1-2): 0
Enter a number (1-2): 2
In a Python program:

- a loop is used to implement 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 = div_sum + divisor
            divisor = divisor + 1

    return div_sum

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