Introduction to Computer Programming

Week 3.1: Loops

The goal of writing a computer program is to automate a process.

Throughout this course, we will study three fundamental topics that underpin automation:

- **Selection:** Decision-making
- **Repetition:** Repeatedly executing a process
- **Modularity:** Chunks of code that can be re-used

What is a loop?

A **loop** is a mechanism that allows the same piece of code to be executed many times

This eliminates the need to copy-and-paste code

Example: Compute the fourth power of a number x:

In [1]:

```
x = 5

ans = x          # first power
ans = ans * x    # second power
ans = ans * x    # third power
ans = ans * x    # fourth power

print(ans)
```

```
625
```

What if we wanted to compute the n-th power of x?

The Fibonacci sequence is a sequence of numbers where each number is the sum of the two preceding ones:

$$f_n = f_{n-1} + f_{n-2}$$

The first two numbers in the Fibonacci sequence are 0 and 1:

$$f_0 = 0, f_1 = 1$$

The sequence starts: 0, 1, 1, 2, 3, 5, 8, ...

In [8]:

```
a = 0
b = 1
c = a + b
print('First Fibonacci is', a)
print('Second Fibonacci is', b)
print('Third Fibonacci is', c)
```

```
First Fibonacci is 0
Second Fibonacci is 1
Third Fibonacci is 1
```

In [7]:

```
a = b
b = c
c = a + b
print('Fourth Fibonacci is', c)

a = b
b = c
c = a + b
print('Fifth Fibonacci is', c)
```

```
Fourth Fibonacci is 2
Fifth Fibonacci is 3
```

Loops in Python

- `for` loops: these repeat code a fixed number of times
- `while` loops: these repeat code until a condition is satisfied

For loops

`for` loops have the syntax:

```
for var in seq:
    # code block (note the indent)
```

The key ingredients are:

1. The keywords `for` and `in`
2. *seq*: an **iterable** *sequence* of values
3. *var*: a variable that takes on each value in *sequence*
4. A colon that follows *sequence*
5. A block of code that is executed at each iteration of the loop. This block of code **must** be indented

Examples using `for` loops

In [10]:

```
for i in [3, 5, 7, 8]:
    print(i)
```

```
3
5
7
8
```

What sequence of events is happening here?

Notice the *sequence* given to the for loop are the number `3, 5, 7, 8` enclosed in brackets `[]`

1. The variable *i* is first assigned the value 3, the first entry in the sequence
2. Then the value of *i* is printed
3. The variable *i* changes to 5, the second entry in the sequence
4. Then the value of *i* is printed again
5. The process repeats until *i* has taken on every value in the sequence

Example: Print the numbers 1 to 5

In [ ]:

Example: Print the numbers 1 to 10 with the help of the `range` function.

`range` takes arguments:

- (optional) start value
- stop value
- (optional) step size

In [ ]:

This exercise will explore the `range` function more

Example: Print the **even** numbers between 0 and 10, in **decreasing** order.

Decreasing: Starts at value other than 0, start (`10`) and stop (`-1`) argument needed

Even: Step size other than 1, step size (`-2`) argument needed

In [ ]:

Example: Loop over a sequence of strings

In [ ]:

Example: Looping mutiple iterables with `zip`

The `zip` function

Input arguments: iterables seperated by commas.

Output: An elementwise series of groups (tuples) containing elements from each iterable.

In [ ]:

The role of the indent

The indent is used to determine which pieces of code are executed in the loop

In [2]:

```
for i in [1, 2, 3]:
    print("I'm in the loop")
print("I'm out of the loop")
```

```
I'm in the loop
I'm in the loop
I'm in the loop
I'm out of the loop
```

The loop involves three iterations, but only the indented code is executed during each iteration

Example: Sum the first five positive integers and print the final value

1 + 2 + 3 + 4 + 5 = 15

In [ ]:

Loops and control flow

Loops often contain `if`, `elif`, `else` statements:

```
for var in sequence:

    if condition:
        # code that is executed if condition == True
    else:
        # code that is executed if condition == False

   # code that is always executed in the loop
```

Extra indents are required for pieces of code that are only executed in the `if` and `else` statements

Example: Print only the even values in the first ten positive integers

In [ ]:

While loops

`while` loops have the syntax

```
while condition:
    # block of code
```

The main components of a while loop are:

1. the keyword `while`
2. *condition*: this is an expression that returns the value True or False
3. an indended block of code that will run as long as *condition* is True

Example of a while loop: Print the numbers from 0 to 4

In [3]:

```
i = 0
while i < 5:
    print(i)
    i += 1
```

```
0
1
2
3
4
```

What sequence of events is happening?

1. The variable *i* is assigned the value of 0
2. The while loop is approached and the condition $i < 5$ is checked
3. Since $0 < 5$ is True, the loop is entered
4. The value of *i* is printed and its value is increased by one
5. The condition $i < 5$ is checked again. Since $1 < 5$ is True, the loop is entered again
6. The process repeats until $i < 5$ is False, at which point the loop is terminated

Which loop to choose?

For loops are used most often when we want to execute a block of code a finite number of times.

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

While loops are used more often when we are unsure exactly how many times we need to execute a block of code before exiting the loop.

As a general rule, choose the loop that minimises the amount of work you need to do/code you need to write.

Example: A square number is an integer of the form n^2.

Print the square numbers, starting from 1, that are smaller than 200.

In [ ]:

Example: Continue requesting numbers from the user until the total input exceeds 100.

In [ ]:

Infinite loops - a word of warning!

Question: What will the output of the following code be?

```
i = 0
while i < 5:
    print(i)
```

Answer: Since the value of i is never changed, the loop will never terminate!

- This is called an **infinite loop**
- One must be careful to avoid these when using `while` loops

Terminating loops using `break`

A `for` or `while` loop can be terminated prematurely using the `break` keyword

In [30]:

```
for i in range(1, 6):

    if i == 3:
        print("Terminating the loop when i = 3")
        break

    print(i)
```

```
1
2
Terminating the loop when i = 3
```

Skipping parts of a loop with `continue`

The `continue` keyword can be used to skip code in a loop

In [31]:

```
for i in range(1, 6):

    if i == 3:
        print("Skipping the case i = 3")
        continue

    print(i)
```

```
1
2
Skipping the case i = 3
4
5
```

When the `continue` keyword is encountered, the current *iteration* of the loop terminates, but the loop continues

For-else

A structure usign for loops that is less often used but can be useful at times.

The indented code after the `else` statement is executed after checking the `if` statement condition `== False` for *all* values iterated through using the for loop.

In [32]:

```
for i in [6, 2, 4]:
    if i%2:
        print("Found odd number")
        break

else:
    print("Didn't find any odd numbers")
```

```
Didn't find any odd numbers
```

Summary

Loops are used to repeatedly execute blocks of code

- `for` loops are used to execute code a certain number of times
- `while` loops are used to execute code until a condition is satisfied
- The `break` keyword will terminate a loop (useful for avoiding **infinite loops**!)
- The `continue` keyword enables blocks of code to be skipped in a loop