Introduction to Computer Programming

Week 3.1: Loops

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
```

Question: what if we wanted to compute the *n*-th power of *x*?

Loops in Python

There are two main 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 sequence:
    # code block (note the indent)
```

The key ingredients are:

1. The keywords `for` and `in`
2. *sequence*: an iterable object such as a list or string
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 [2]:
```
for i in [3, 5, 7, 8]:
    print(i)
```

```
3
5
7
8
```

What sequence of events is happening here?

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

```
1
2
3
4
5
```

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

In [4]:
```
for n in range(1, 11):
    print(n)
```

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

The exercises will explore the `range` function more

Example: Print the squares of the first five (positive) integers

In [5]:
```
for n in range(1,6):
    print(n**2)
```

```
1
4
9
16
25
```

Example: Loop over a list of strings

In [3]:
```
cities = ['Toronto', 'Barcelona', 'London']
for c in cities:
    print("I'd like to be in", c)
```

```
I'd like to be in Toronto
I'd like to be in Barcelona
I'd like to be in London
```

Example: Looping with `zip`

In [4]:
```
cities = ['Toronto', 'Barcelona', 'London']
seasons = ['summer', 'spring', 'summer']
for c,s in zip(cities, seasons):
    print("I'd like to be in", c, "in the", s)
```

```
I'd like to be in Toronto in the summer
I'd like to be in Barcelona in the spring
I'd like to be in London in the summer
```

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 integers and print the final value

In [9]:
```
sum = 0
for i in range(1,6):
    sum += i
print(sum)
```

```
15
```

Loops and control flow

Loops commonly contain `if` 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 the first few even integers

In [3]:
```
for i in range(1,10):

    if i % 2 == 0:
        print(i)
```

```
2
4
6
8
```

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 [13]:
```
i = 0
while i < 5:
    print(i)
    i += 1
```

```
0
1
2
3
4
```

What sequence of events is happening in the previous example?

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

Example: A square number is an integer of the form n^2. Print the square numbers that are smaller than 150.

In [4]:
```
n = 1
while n**2 < 150:
    print(n**2)
    n += 1
```

```
1
4
9
16
25
36
49
64
81
100
121
144
```

Example: Looping over entries of a list with a `while` loop

In [12]:
```
cities = ['Toronto', 'Barcelona', 'London']
i = 0
while i < len(cities):
    print(cities[i])
    i += 1
```

```
Toronto
Barcelona
London
```

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 [3]:
```
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 [4]:
```
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

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