

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:

- The keywords `for` and `in`
- sequence*: an iterable object such as a list or string
- var*: a variable that takes on each value in *sequence*
- A colon that follows *sequence*
- 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 [1]: for i in [3, 5, 7, 8]:
        print(i)

3
5
7
8
```

What sequence of events is happening here?

- The variable i is first assigned the value 3, the first entry in the sequence
- Then the value of i is printed
- The variable i changes to 5, the second entry in the sequence
- Then the value of i is printed again
- 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.

```
In [ ]:
```

The exercises will explore the `range` function more

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

```
In [ ]:
```

Example: Loop over a list of strings

```
In [ ]:
```

Example: Looping with `zip`

```
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 integers and print the final value

```
In [ ]:
```

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 [ ]:
```

While loops

`while` loops have the syntax

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

The main components of a while loop are:

- the keyword `while`
- condition*: this is an expression that returns the value `True` or `False`
- an indented 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 in the previous example?

- The variable i is assigned the value of 0
- The while loop is approached and the condition $i < 5$ is checked
- Since $0 < 5$ is `True`, the loop is entered
- The value of i is printed and its value is increased by one
- The condition $i < 5$ is checked again. Since $1 < 5$ is `True`, the loop is entered again
- 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 [ ]:
```

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

```
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 [2]: 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 [1]: 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