Week 2
Types, lists & functions

Data types

If you want to write a program that allows the user to input something, you can use the command `input`:

```
name = input("What is your name? ")
print("Hello "+name)
```

If you want the user to enter a number, however, the following code will lead to a `TypeError`:

```
c = input("Enter a temperature in Celsius: ")
print("In Fahrenheit, this is",9*c/5+32)
```

In Python, variables all have a type. Without realising it, you have already met at least five types:

1. `int` – An integer, for example `a = 1`
2. `complex` – A complex number, for example `a = 1+2j`
3. `float` – An floating point number (decimal), for example `b = 1/2`
4. `str` – A string of characters, for example `c = "Hello"`
5. `bool` – A boolean, for example `d = True`

Operator such at `+`, `-` and `*` will behave differently for different combinations of types:

```
print(10*2)
print("10"*2)
print(10+2)
print("10"+"2")
```

```
20
1010
12
102
```

You can check the type of a variable by using the function `type`:

```
a = 10
print(type(a))
print(type("Hello"))
```

```
<class 'int'>
<class 'str'>
```

`input` will always return what the user types as a `str`, so to make the example above work, you must convert the input to an `int` (or a `float` or `complex`):

```
c = int(input("Enter a temperature in Celsius: "))
print("In Fahrenheit, this is",9*c/5+32)
```

1. In the editor window, type the following code and replace each `$` with one of `a`, `b`, `c`, or `d`.

```
a = 5
b = "5"
c = 2
d = 2.0
print($ * $ + $)
```

What do you need to replace each `$` with to make Python print:

a. 555 **b.** 15 **c.** 15.0 **d.** `TypeError`

Lists

As the name suggests, a list is a list of things. They are very useful in Python for storing multiple pieces of data. Lists are created using square brackets:

```
a = [4,10,3,7,0,-10]
b = []
c = [4,"Hello",True,8]
```

Items in a list can be obtained by writing an index in square brackets after the list's name:

```
a = [4,10,3,7,0,-10]
print(a[2])
print(a[1])
print(a[0])
print(a[-1])
print(a[-2])
```

```
3
10
4
-10
0
```

The numbering of a list starts at 0 (leftmost item). A negative index of `-i` will return the `i`th item from the right of the list (-1 is the rightmost item).

You can also change the value of an item in a list:

```
a = [4,10,3,7,0,-10]
a[2] = 8
print(a)
```

```
[4, 10, 8, 7, 0, -10]
```

If you want to get multiple items from a list, you can take a slice of a list:

- `list[a:b]` will get the items in the list from `a` up to and not including `b`
- `list[:b]` will get the items in the list from the start up to and not including `b`
- `list[a:]` will get the items in the list from `a` up to the end
- `list[a:b:c]` will get every `c`th item in the list from `a` up to and not including `b`

```
a = [4,10,3,7,0,-10]
print(a[1:3])
print(a[4:])
print(a[:-3])
print(a[1:5:2])
print(a[:5:2])
print(a[::2])
```

```
[10, 3]
[0, -10]
[4, 10, 3]
[10, 7]
[4, 3, 0]
[4, 3, 0]
```

The same syntax for getting items and slices from a list also works for strings:

```
words = "Hello there!"
print(words[2])
print(words[-1])
print(words[3:10])
print(words[3:10:2])
print(words[::-1])
```

```
l
!
lo ther
l hr
!ereht olleH
```

2. In the editor window, type the following code and replace the `$` with something.

```
a = [4,10,3,7,0,-10]
print(a[$])
```

What do you need to replace the `$` with to make Python print the following. There are multiple possible answers for each part.

a. 4	**c.** -10	**e.** `[4,10]`	**g.** `IndexError`
b. 10	**d.** 0	**f.** `[ ]`	**h.** `TypeError`

You can add items to the end of a list using the method `append` and you can concatenate two lists by using the `+` operator:

```
a = [4,10]
a.append(5)
print(a)
b = [1,5]
a += b
print(a)
print(b+b)
```

```
[4, 10, 5]
[4, 10, 5, 1, 5]
[1, 5, 1, 5]
```

There are some useful inbuilt functions that can give you some information about a list. The following example show how to find the maximum and minimum elements in a list, the sum of all the element and the length (number of elements in) of the list:

```
a = [4,10,3,7,0,-10]
print(max(a))
print(min(a))
print(sum(a))
print(len(a))
```

```
10
-10
14
6
```

3. How could you get Python to print the mean of a list?

4. Write a Python script that generates a list containing all the square numbers between 1 and 100.

There are a few more useful methods of a list. `count` tells you how many times something appears in a list. `sort` will sort the list into order.

```
a = [4,10,3,10,0,10]
print(a.count(10))
print(a.count(0))
a.sort()
print(a)
```

```
3
1
[0, 3, 4, 10, 10, 10]
```

You can use a `for` loop to do something for every element in a list:

```
for i in [3,2,4,10]:
    print(i**2)
```

```
9
4
16
100
```

Sometimes, you may need to make a `for` loop go through both the entries in the list and their indices. You can use `enumerate` to do this:

```
a = [2,3,5,7,11]
for i,j in enumerate(a):
    print(i)
    print(j)
```

```
0
2
1
3
2
5
3
7
4
11
```

5. Write a Python script that prints the remainders when the square numbers between 1 and 100 are divided by 5. (Hint use the script you wrote for the previous question.)

6. The following code is incomplete.

```
a = [1, 3, 2]
b = [2, 1, -1]

dot = 0
for i in range(3):
    pass
print(dot)
```

Replace `pass` with some Python code so that the dot product of the vectors `a` and `b` is calculated and printed.

Dictionaries

A list is a way of storing multiple items in order, numbered 0, 1, 2, ... Sometimes it is more useful to store multiple items not in order, with each one identified by a `key`. This can be done by storing the items in a dictionary.

Dictionaries are created using curly brackets and behave similarly to lists:

```
a = {"cat":5, "dog":10, "rabbit":True, 5:"apple"}
print(a["cat"])
print(a[5])
a["cat"] += 1
a["hat"] = -1
```

```
5
apple
```

The items in the dictionary each have a `key` and a `value`, and are written in the form `key : value`.

You can use a for loop to go through each key, each value, or both:

```
data = {"hat":1, "scarf":2, "coat":1, "shoes":5}
for i in data: # goes through keys
    print(i)
```

```
hat
scarf
coat
shoes
```

```
for i in data.keys(): # goes through keys
    print(i)
```

```
hat
scarf
coat
shoes
```

```
for i in data.values(): # goes through values
    print(i)
```

```
1
2
1
5
```

```
for i,j in data.items(): # goes through both
    print(i)
    print(j)
```

```
hat
1
scarf
2
coat
1
shoes
5
```

7. The following dictionary gives the marks out of 100 that six fictional students scored in a test.

```
results = {"Anna":51, "Ben":83, "Chris":12, "Diana":47, "Elizabeth":84}
```

A score of more than (or equal to) 80 is a distinction. A score of more than (or equal to) 40 (and less than 80) is a pass. A score of below 40 is a fail.

Write a Python script that prints the students' names and whether they got a distinction, passed or failed.

Tuples

A tuple is like a list, but it's entries cannot be changed or added to after it is made. Tuples are created using regular brackets:

```
a = (2,3,10)
b = (1,)
c = (,)
```

To make a tuple with 1 or 0 entries, you must include the comma.

Loading lists, dictionaries, and tuples from files

You can read and write lists, dictionaries, and tuples from and to files using the `json` module. This is useful if you ever build a list (or dictionary or tuple) that takes a long time to make and you want to use again without having to build it.

To save a list, do the following:

```
import json
data = [2,3,5,7,11,13]
with open("data.json","w") as f:
    json.dump(data, f)
```

To load the list, do the following:

```
import json
with open("data.json") as f:
    data = json.load(f)
```

Functions

If you want to use a piece of code multiple times on different numbers, you can define a function in Python.

```
def f(x):
    return x**2
```

`def` is short for define, and tells Python you are about to define a function. This is followed by `f`, the name of the function. In brackets, we give the input(s) of the function: this function will take one input that it will call `x`. The command `return` will cause the function to give `x**2` as its output, then stop running.

This function can then be used as in the following example code.

```
def f(x):
    return x**2

print(f(4))
a = f(2)
print(a)
```

```
16
4
```

Functions can take more than one input as in the following example. They also do not have to return anything.

```
def multiply_and_print(a, b):
    print(a*b)
```

Functions can contain multiple lines of code. It is common to write a string that describes what the function does as the first line of the function. If `help` is called on the function, it will display this text.

```
def product(a):
    "Returns the product of all the items in a list"
    output = 1
    for i in a:
        output *= i
    return output

help(product)
```

```
Help on function product in module __main__:

product(a)
    Returns the product of all the items in a list
```

8. Using the function above, print the product of the prime numbers up to 100.

Putting it all together

Using all the Python commands you have learned in the two weeks so far, do the following questions.

9. It is known that

$$\ln 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \frac{1}{7} - \ldots$$

Write a Python script to add up the first 1000 terms in this series by appending every term to a list, then using the `sum` function.

10. Write a Python function that takes two numbers as an input and returns their highest common factor. Write a second function that takes two numbers as an input and returns their lowest common multiple.

11. Write some Python code that makes a list of the first 20 triangle numbers. Print the 20th triangle number. Print the sum of the first 20 triangle numbers?

(Reminder: The nth triangle number is $1 + 2 + 3 + ... + n$.)

12. Write a Python script that generates a list containing all the prime numbers between 1 and 100. Prints the squares of all the prime numbers between 1 and 100.

13. There is a row of 1001 open lockers, numbered from 0 to 1000. One person walks along the row of lockers and closes all the even numbered lockers. A second person walks along and changes the state (opens if they're closed; closes if they're open) of all the lockers that are multiples of 3. More people do the same for multiples of 4, then 5, then 6, and so on until the final person changes the state of the multiples of 1000.

Which lockers are closed at the end?

Hint: You can use the following Python code to make a list containing `True` 1001 times (ie for 0 to 1000 inclusive):

```
lockers = [True]*1001
```

14. Write a function that returns the sum of all the factors of a number (including 1 but excluding the number itself).

A perfect number is a number (such as 6) that is equal to the sum of its factors. Use your function to find the next three perfect numbers after 6.

15. A pair of numbers a and b are called amicable if a is the sum of the factors of b and b is the sum of the factors of a. Use the function you wrote in the previous section to find pairs of amicable numbers below 500.

Assessed question

When you have completed this question, you should show your code to one of the helpers in your class.

16. A function $f : \mathbb{N} \to \mathbb{N}$ is defined by

$$f(n) = \begin{cases} \dfrac{n}{2} & \text{if } n \text{ is even} \\ 3n + 1 & \text{if } n \text{ is odd} \end{cases}$$

Picking a starting number and repeatedly applying this function is conjectured to always eventually lead to the number 1. (This is called the Collatz conjecture.)

27 requires 111 applications of f before reaching 1, and is the lowest number that takes more than 100 applications of f to reach 1.

What is the lowest number that takes more than 200 applications of f to reach 1?