Introduction to Computer Programming

Objects and Variables

Objects

- Every item of data (numbers, text characters etc) in a Python program can be described by the term **object**
- The type of an object determines what properties it has and how it can be used in the Python program

In [2]:
```
30
'Python'
1.2
```

Out[2]:
```
1.2
```

Variables and Variable Assignment

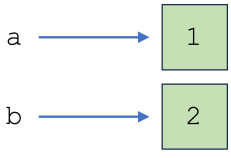
- A variable is a name that refers or points to a particular object
- By *assigning* an object to a variable, we allow it to be manipulated within the program, using the variable name
- To create a variable, we simply assign it a value
- Assignment is achieved with a single equals sign (=)

In [12]:
```
b = 4

print(b)
```

```
4
```

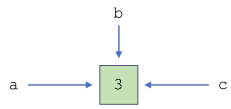
The Python function `print()` displays whatever is between the parentheses `(...)`



In [16]:
```
a, b = 1, 2

print(a, b)
```

```
1 2
```



In [18]:
```
a = b = c = 3

print(a, b, c)
```

```
3 3 3
```

Object Types

The type of an object determines what properties it has and how it can be used in the Python program

Basic Object Types
(not exhaustive)

- `int` **integer**: (e.g. 3, 88)
- `float` **floating point number**: number with decimal point (e.g. 1.5, 99.9626)
- `str` **string**: text data enclosed within quotation marks (e.g. `'hello'` or `"12"`)
- `bool` **Boolean**: `True` or `False` (first letter capitalised)

Integer (`int`)

A whole number, with no decimal place

In [45]:
```
print(type(1))
```

Out[45]:
```
int
```

The Python function `type()` returns the type of an object, or variable that points to an object, within the parentheses `(...)`

The Python function `print()` displays whatever is between the parentheses `(...)`

In [21]:
```
a = 1
print(a)
print(type(a))
```

```
1
<class 'int'>
```

Floating point (`float`)

A fractional number or a decimal number (consisting of a whole and a fractional part)

In [22]:
```
print(type(1.0))
```

Out[22]:
```
float
```

In [24]:
```
b = 1.0
print(b)
print(type(b))
```

```
1.0
<class 'float'>
```

Numerical value storage

Decimal numbers

In daily life we mostly use the **decimal** or **base-10** number system.

Each number is formed by finding the sum of base-10 numbers, each multiplied by a factor (0-9).

Consider the number 104.02

Unit	10^2	10^1	10^0	10^{-1}	10^{-2}
Value	100	10	1	0.1	0.01
Factor	1	0	4	0	2
Total	100	0	4	0	0.02

Sum of columns = 100 + 4 + 0.02 = 104.02

Binary numbers

Numbers are stored on a computer as **binary** or **base-2** numbers.

Each number is formed by finding sum of base-2 numbers, each multiplied by a factor **1 or 0**.

The factors are like "on" (1) and "off" (0) switches for the different columns

Consider the number 25

Unit	2^4	2^3	2^2	2^1	2^0
Value	16	8	4	2	1
Factor	1	1	0	0	1
Total	16	8	0	0	1

Sum of columns (bits) = 16 + 8 + 1 = 25

A bit is the smallest unit of data that a computer can process and store

Integer storage

Itegers are stored as **signed** binary numbers.

Signed means we can store both negative and positive values.

For a given number of bits, the most significent bit (m.s.b) is negative.

Consider the number - 10, represented using 5 columns (bits)

Unit	-2^4	2^3	2^2	2^1	2^0
Value	- 16	8	4	2	1
Factor	1	0	1	1	0
Total	-16	0	4	2	0

Sum of columns (bits) = -16 + 4 + 2 = -10

Any integer (whole number) can be represented in binary form (if we have enough units/bits).

Integer storage in Python does not place a limit on the number of bits that can be used to store a number.

The upper limit on available bits to store an integer is determined by the amount of memory a computer system has.

Fixed point storage

To store fractional decimal numbers in binary, base-2 numbers with **negative exponents** are used.

Consider the number 4.5

	2^2	2^1	2^0	2^{-1}	2^{-2}
Value	4	2	1	0.5	0.25
Factor	1	0	0	1	0
Total	4	0	0	0.5	0

We can represent 4.5 exactly using the 5 columns shown here (5 bits).
Sum of columns = 4 + 0.5 = 4.5

Now consider the number 4.125

	2^2	2^1	2^0	2^{-1}	2^{-2}
Value	4	2	1	0.5	0.25
Factor	1	0	0	0	1
Total	4	0	0	0	0.25

We *can't* represent 4.125 exactly using 5 columns (5 bits)

Closest estimate: 4.25 (shown in the table) or 4
(each result contains an error of 0.125)

If we had an extra column (bit), 2^{-3} (0.125), we could represent the number exactly.

Python does not place a limit on the number of bits that can be used to store an **integer**.

However, in Python (and many other programming languages) **fractional numbers** are stored on a computer using a fixed number of 64 bits

This means we must use a *finite* number of bits to try and represent an *infinite* amount of numbers.

Therefore many fractional floating point numbers can't be represented exactly when stored in binary form.

Floating point numbers

Floating point number storage is the system used to store binary numbers on a computer.

A thorough explanation of binary floating point is complex and beyond the scope of this class (but if you are curious, there is a link to a full explanation on the final slide).

However, **floating point** number storage is similar to *scientific notation* for decimal (base-10) values.

$$1.234 \times 10^{0} = 1.234$$
$$1.234 \times 10^{1} = 12.34$$
$$1.234 \times 10^{2} = 123.4$$

$$\underbrace{M}_{mantissa} \times \underbrace{10}_{base}{}^{\overbrace{E}^{exponent}}$$

$$1.234 \times 10^{0} = 1.234$$
$$1.234 \times 10^{1} = 12.34$$
$$1.234 \times 10^{2} = 123.4$$

The **mantissa** controls the *precision* of a number.
(normalised so that a maximum of 1 significant figure (s.f.) is to the left of the decimal point)

The **exponent** controls the *order of magnitude* of the number.

If we limit the number of bits in the mantissa (e.g. 4 bits, all positive) and the number of bits in the exponent (e.g. 1 bit, positive), there are numbers we can't represent.

Examples (exponent too high / not enough bits to store number in mantissa):
1.5×10^{10}
1×10^{-4} (or 0.0001)

$$\underbrace{M}_{mantissa} \times \underbrace{10}_{base}^{\overbrace{E}^{exponent}}$$

$$1.234 \times 10^{0} = 1.234$$
$$1.234 \times 10^{1} = 12.34$$
$$1.234 \times 10^{2} = 123.4$$

Binary **floating point** number storage is similar to *scientific notation* for decimal (base-10) values

However binary floating point numbers have a *binary* mantissa and *binary* exponent, 53 and 11 bits respectively in 64 bit 'double precision' format.
(link to a full explanation on the final slide)

What does 'floating point' mean?

$$\underbrace{M}_{mantissa} \times \underbrace{10}_{base}^{\overbrace{E}^{exponent}}$$

$$1.234 \times 10^{0} = 1.234$$
$$1.234 \times 10^{1} = 12.34$$
$$1.234 \times 10^{2} = 123.4$$

The exponent means that the decimal point can 'float' to different positions within the number, which is where the term 'floating point' comes from.

Floating point error

The main concept to grasp is that the fixed number of bits used to represent a floating point number can lead to error in the stored represetation of the number.

When the Python function `print()` displays a value, the number of decimal places displayed is determined by the format of the value entered within the parentheses.

In [6]:
```
print(0.1)
```

```
0.1
```

The Python function `format()` displays the first value within the parentheses formatted using the second value within the parentheses.
`'.17f'` : display a floating point number to 17 decimal places.

In [26]:
```
format(0.1, '.17f') # value to 17 dp
```

Out[26]:
```
'0.10000000000000001'
```

We can see that the stored value of `0.1` contains an error, which we refer to as **floating point error**.

However, because we have 64 bits to store the number, the difference between the true value and the stored value of the floating point number is small (in the order of 17 d.p. in this example).

In [7]:
```
print(0.2)

format(0.2, '.17f')
```

```
0.2
```

Out[7]:
```
'0.20000000000000001'
```

In [29]:
```
print(0.3)

format(0.3, '.17f')
```

```
0.3
```

Out[29]:
```
'0.29999999999999999'
```

String (`str`)

A collection of characters (alphabetical, numerical, or other e.g. punctuation)

In [30]:
```
c = "10"
d = 'python 3'
e = 'hello world!'

print(type(c))
print(type(d))
print(type(e))
```

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

A collection of characters is a string if it is enclosed within single (`'...'`) or double (`"..."`) quotation marks.

Unlike numerical data, `strings` are **subscriptable**

This means that each character of the string has an **index** that can be used to access it

Characters are **indexed** with integer values, starting from 0

We can return the Nth character of a string with `string_name[N]`

We can return the characters between element N and element M with `string_name[N:M]`

Print the first letter of x

In [31]:
```
x = 'Hello'

print(x[0])
```

```
H
```

Print the last letter of x

In [34]:
```
print(x[4])
print(x[-1])
```

```
o
o
```

Print the first 3 letters of x

In [37]:
```
x = 'Hello'
print(x[0:3])
print(x[:3])
```

```
Hel
Hel
```

Print the last 3 letters of x

In [44]:
```
print(x[2:5])
print(x[-3:])
```

```
llo
llo
```

There are a number of operations that can be performed on a `string`

These are known as **methods**

The method is used on a variable by placing the method name after the variable name, seperated by a dot/period/point `.`

A full list of string methods can be found here:
https://www.w3schools.com/python/python_ref_string.asp (https://www.w3schools.com/python/python_ref_string.asp)

Examples:

- `upper` (converts string to upper case letters)
- `find` (finds index of first ocurance of a specific character)

In [42]:
```
x = 'hello'
y = x.upper()
print(y)
```

```
HELLO
```

In [43]:
```
z = x.find('l')
print(z)
```

```
2
```

Boolean (`bool`)

Binary True or False values

In [4]:
```
f = True
g = False

print(type(f))
print(type(g))
```

```
True <class 'bool'>
```

Need to see some more examples?

https://www.w3schools.com/python/python_variables.asp (https://www.w3schools.com/python/python_variables.asp)
https://www.geeksforgeeks.org/python-variables/ (https://www.geeksforgeeks.org/python-variables/)

Want to take a quiz?

https://realpython.com/quizzes/python-variables/ (https://realpython.com/quizzes/python-variables/)
https://pynative.com/python-variables-and-data-types-quiz/ (https://pynative.com/python-variables-and-data-types-quiz/)

Want some more advanced information?

Objects and Types
https://realpython.com/python-data-types/ (https://realpython.com/python-data-types/)
https://realpython.com/python-variables/ (https://realpython.com/python-variables/)
https://pynative.com/python-variables/ (https://pynative.com/python-variables/)

Floating point numbers
https://bteccomputing.co.uk/use-of-binary-to-represent-negative-and-floating-point-numbers/ (https://bteccomputing.co.uk/use-of-binary-to-represent-negative-and-floating-point-numbers/)
https://www.youtube.com/watch?v=L8OYx1I8qNg (https://www.youtube.com/watch?v=L8OYx1I8qNg)

In [ ]: