Python Data-Types

Relevant for: COMP5711M

Brandon Bennett

School of Computing
University of Leeds

B.Bennett@leeds.ac.uk

Last Updated: August 2019

Learning Goals

- Appreciate the wide variety of data that is involved in computer applications.
- Be aware of the basic data types provided by Python.
- Understand that for many purposes it is useful to make use of complex data types, such as lists and dictionaries.

Data and its Types

The word *data* refers to any kind of information that is stored in some specific format.

Nearly all computer programs involve some kind of data processing.

This data can vary greatly in its type, quantity, complexity.

And there are a huge number of ways that a computer program can operate with data: it may analyse it to extract useful new information; it may manipulate it to convert it into new forms.

Data Examples

- 3 numbers describing the size of a box,
- a sequence of temperature measurements,
- financial information (e.g. bank accounts),
- a database of information about employees of a company,
- an inventory of products and stock held by a supermarket,
- a 3D representation of a human body,
- the text of a book,
- audio data (e.g. in mp3 or flac format),

- image data (e.g. photos in jpeg format)
- video data (e.g. in mpeg format),
- a large repository of URLs and textual data harvested from the web.

Basic Types

Python uses the following basic data types:

- Numbers (integers and floats)
- Booleans
- Strings
- Lists
- Tuples
- Dictionaries

This lecture will describe the values and common operations of the various Number types and Booleans.
The other types will be covered in detail later.

Number Types

int — int can hold integer valued number of unspecified length e.g. -17, -1, 0, 2, 5, 10, 42, 15647989

float — float can hold floating point number allowing the approximate representation of real numbers.

complex — complex can hold a complex number.

Number Type Conversion

Unlike some languages, will automatically convert between types when this is intuitively appropriate.

For instance, when we divide one integer by another, we may get a fraction as the result, and this will be represented by a float.

```
>>> x = 3
>>> type(x)
<class 'int'>
>>> y = 5
>>> z = x/y
>>> z
0.6
>>> type(z)
<class 'float'>
```

Mathematical Operators

Operator	Operation
`x + y`	Addition of `x` and `y`
`x - y`	Subtraction of `y` from `x`
`x * y`	Multiplication of `x` and `y`
`x / y`	True division of `x` by `y`. The result of this operation is always a floating point number
`x // y`	Floor division of `x` by `y`. The result of this operation is the integer part of the result of the division
`x % y`	Modulo. The result of this operation is the remainder when `x` is divided by `y`
`-x`	Negation of `x`
`x**y`	Has the value of `x` raised to the power of `y` — i.e. x^y.

Booleans

Like many programming languages Python provides the Boolean type (named after the logician George Boole), which has only two possible values:

- `True`
- `False`

These values can be used directly, or assigned to variables, but more often they occur as the values of *test* operations. For example, comparisons such as:

- `3 > 2`
- `1 + 1 == 2`

- `input_string == "yes"`

Boolean Operators

Boolean values can be manipulated by means of the Boolean operators: `and`, `or` and `not`:

- `not` *test* is `True` if *test* is `False` and `False` if *test* is `True`.
- *test1* `and` *test2* is `True` if both *test1* and *test2* are `True`.
- *test1* `or` *test2* is `True` if either *test1* or *test2* is `True`.

For example:

```
x > 7 and x < 52
```

```
s == "yes" or (s == "no" and (not h > 42))
```

Strings, Lists, Tuples and Dictionaries

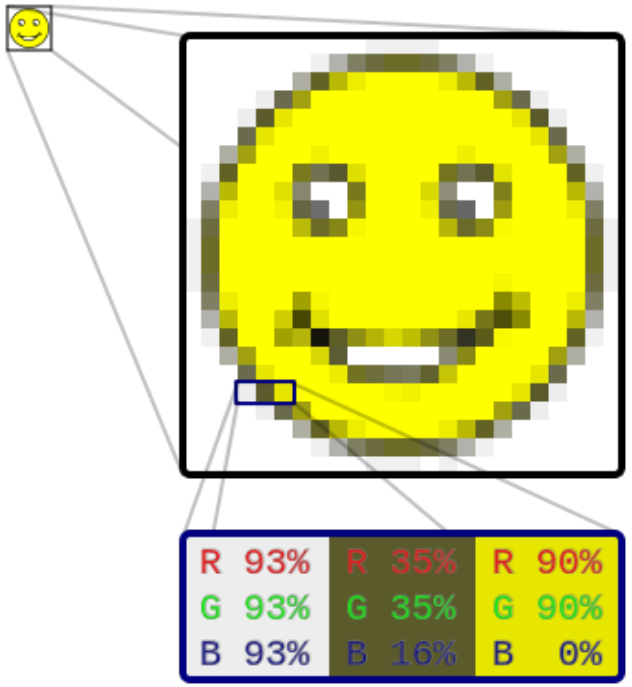
Here are some illustrations of these types.
They will be covered in detail later:

```
my_name = "Brandon Bennett"
list_of_numbers = [1, 3, 7, 9, 12, 10, 2]
lixt_of_strings = ["this", "is", "a", "list"]
nested_list = [ [1,2,3], [4,5,6], [7,8,9] ]
character = [ "Bob", 27,
              ["items", ["gun", "cake", "onion"]]
            ]
grades =  { "Asterix'':57, "Bibs":72,
            "Cazzgr":45,   "Dilbert":3 }
```

Images

There are many ways to represent images.

A common and straightforward representation is to use a 2-dimensional array of pixels, each of which is associated with 3 numbers representing the intensity of red, green and blue light at that pixel.



In Python, a 2-dimensional array is represented by a list of lists.

The 6th pixel along of the 20th row down, might be set to the colour yellow as follows:

```
image[20][6] = (90,90,0)
```

Classes and Objects

Later in the module you will learn about the *object oriented* features of Python.

A *class* defines a complex data type that may be associated with a combination of several simpler pieces of data. (It may also define *methods* that specify operations on that data.)

An *object* is a particular instance of a class.

```
class Tile:
    def __init__(self, w, h, c):
        self.width = w
        self.height = h
        self.colour = c
        self.material = "ceramic"

my_tile = Tile( 10, 10, "red" )  ## an object of class Tile
```