

CMM201 Week 3: Data Structures in Python

1 Introduction to Data Types and Data Structures

1.1 Aims of the Module

- Learn and understand the different data types and data structures contained in Python.
- Exemplify practical uses of each type and structure.

1.2 Additional Reading and Sources

- [Real Python](#)
- [Data Flair](#)

2 Number Types

- Integers
- Booleans
- Float

2.1 Integers

- The most basic data type in Python.
- A number by default is an integer if no decimal value is specified.
- The `type()` function can be used to discover the type of a variable or a number.
- You can use comparison operators to evaluate integer values.

```
In [44]: type(3)
```

```
Out[44]: int
```

```
In [10]: ## Writing exponential numbers  
         2e5
```

```
Out[10]: 200000.0
```

```
In [50]: x=3  
         print(type(x))
```

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

```
In [29]: # See if two ints are equal
        3==5
```

```
Out[29]: True
```

```
In [31]: # See if two ints are not equal
        3!=5
```

```
Out[31]: True
```

```
In [65]: # See if a number is smaller than another
        3<5
```

```
Out[65]: True
```

```
In [66]: # See if a number is smaller or equal to another
        3<=5
```

```
Out[66]: True
```

```
In [68]: # See if a number is larger than another
        3>5
```

```
Out[68]: False
```

```
In [69]: # See if a number is larger or equal to another
        ##3>=5
```

```
Out[69]: False
```

2.2 Booleans (logical operators)

- Notice that we have obtained *True* and *False* as results.
- These results are also data, and they belong to the *boolean* (bool in Python) type.

```
In [22]: type(True)
```

```
Out[22]: bool
```

```
In [70]: z = True
        print(type(z))
```

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

2.3 Float

- Is how we call decimals in Python.
- Up to 15 decimal places.

```
In [5]: y=6.91289739812784749872987
```

```
In [6]: type(y)
```

```
Out[6]: float
```

```
In [20]: # If more than 15 decimal places are used, python truncates  
         7.0789349236894739847398972348974238947
```

```
Out[20]: 7.078934923689474
```

```
In [32]: # We can also compare floats, but funny things may happen!  
         1.1+2.2==3.3
```

```
Out[32]: False
```

Do you know why?

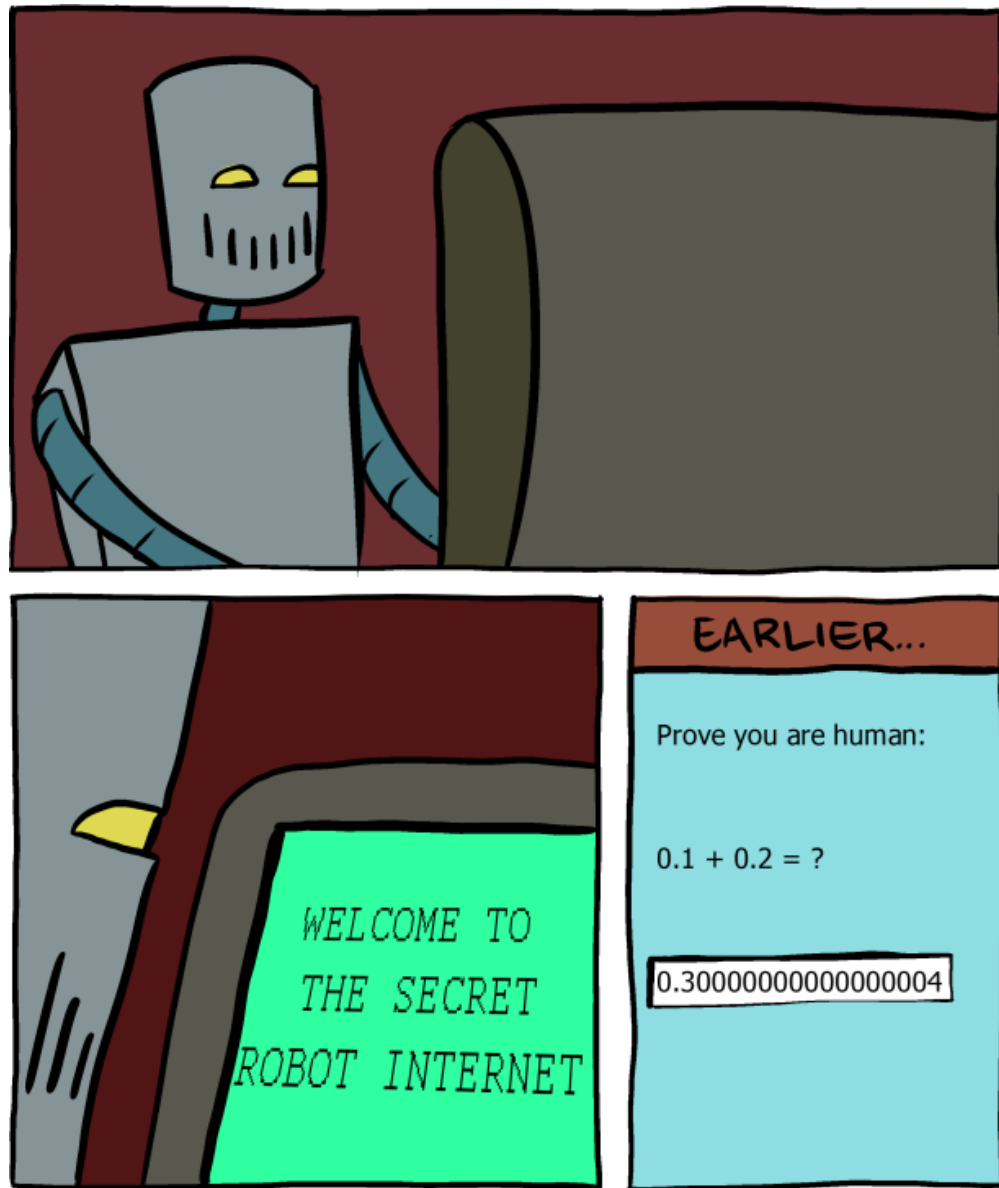


Fig. 2. Prove you are a robot!

2.4 The *isinstance()* function

- Now that we know (at least) two data types, we can see how this function works.
- The function takes two *inputs/arguments*, the variable containing the data and the type to be tested.
- Let's see if *x* and *y* contain integers.

```
In [21]: isinstance(x,int)
```

```
Out[21]: True
```

```
In [7]: isinstance(y,int)
```

```
Out[7]: False
```

You can also work with complex, binary, octal and hexadecimal numbers in Python, but no need to stress! (this is not a maths class...)

2.5 Strings

- You can store letters and words in a variable.
- Strings are defined using quotations (single or double).

```
In [34]: a = 'Hi'
         print(a)
```

```
Hi
```

```
In [48]: # Any number inside quotations becomes a string
         c = "45"
         print(c)
```

```
45
```

```
In [47]: # A boolean inside quotations becomes a string
         b = "True"
         print(b)
```

```
True
```

```
In [49]: print(type(a),type(a),type(a))

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

3 Conversion Functions

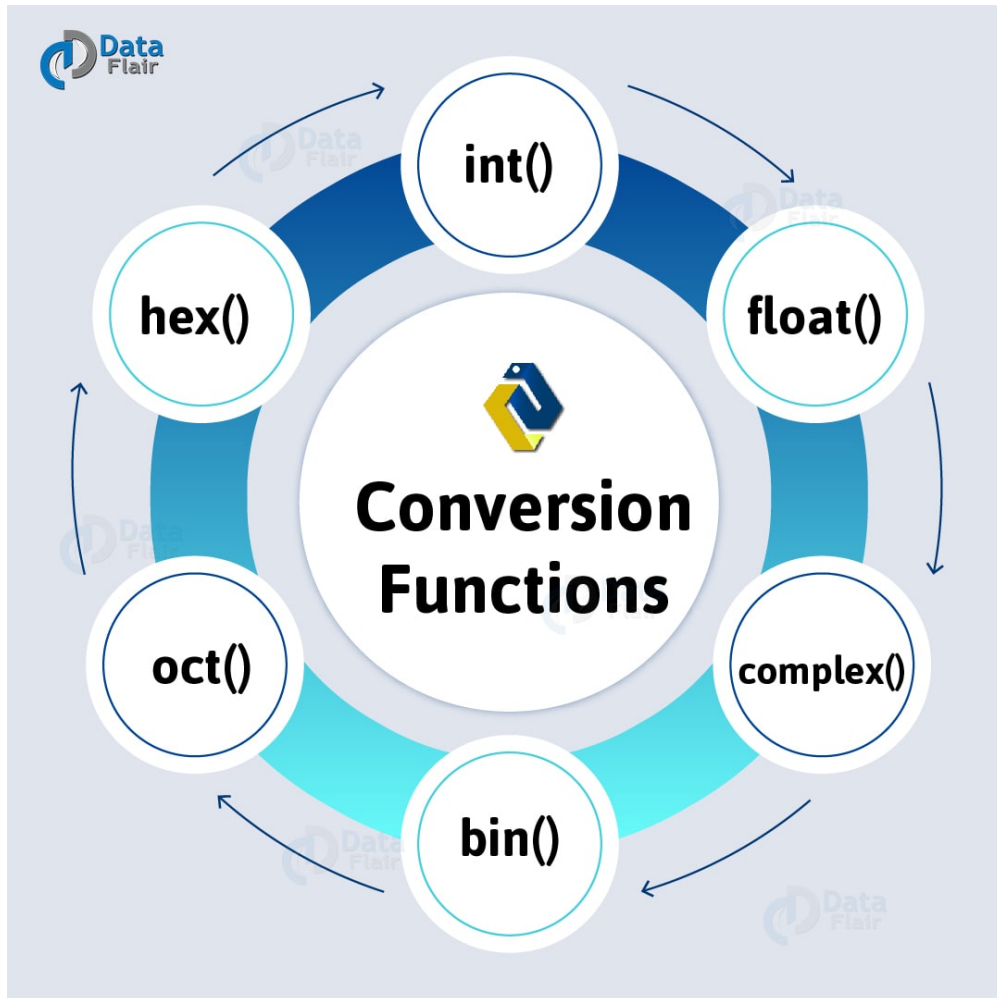


Fig. 3. Conversion Functions

```
In [24]: # the int() function can convert any number into an integer.  
# Mostly used to round floats  
int(7.8)
```

```
Out[24]: 7
```

```
In [13]: # Converting booleans into ints  
int(True)
```

```
Out[13]: 1
```

```
In [14]: # float function adds a 0 decimal to an int  
float(9)
```

```
Out[14]: 9.0
```

```
In [15]: # bool() turns any number to TRUE (other than 0)
         bool(8)
```

```
Out[15]: True
```

```
In [16]: bool(0)
```

```
Out[16]: False
```

4 Data Structures

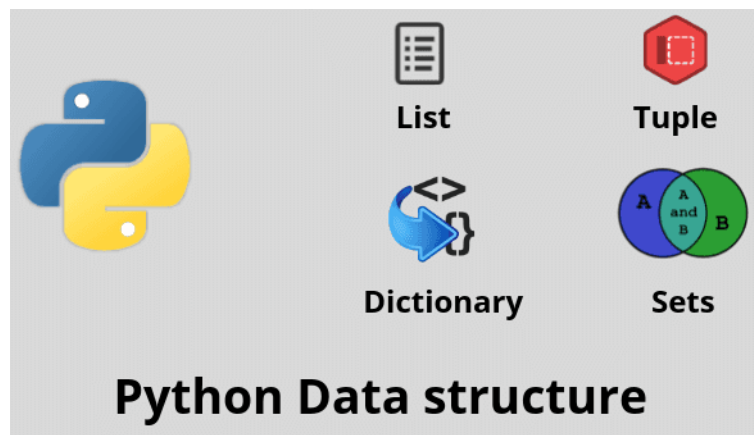


Fig. 4. Python Data Structures

4.1 Tuples

- **IMMUTABLE** collection of elements.
- Defined using parenthesis and separating elements with commas.
- Not all elements in a tuple have to be of the same type.

```
In [52]: tuple1 = (1,2,3)
         tuple1
```

```
Out[52]: (1, 2, 3)
```

```
In [57]: tuple2 = (1, 'g', 4.4, True)
         tuple2
```

```
Out[57]: (1, 'g', 4.4, True)
```

4.2 Lists

- **MUTABLE** collection of elements.
- Defined using squared brackets and separating elements with commas.
- Not all elements in a tuple have to be of the same type.

```
In [54]: list1 = [1,2,3]
        list1
```

```
Out[54]: [1, 2, 3]
```

```
In [59]: list2 = [1,'g',4.4,True]
        list2
```

```
Out[59]: [1, 'g', 4.4, True]
```

4.3 The `len()` function

- We can learn how many elements are contained in a tuple or in a list by using this function.

```
In [56]: len(tuple1)
```

```
Out[56]: 3
```

```
In [60]: len(list2)
```

```
Out[60]: 4
```

4.4 Accessing an element in a tuple/list

- We can access to all positions in a tuple/list by using squared brackets **after** the tuple/list.
- **Indexes in Python begin in 0!**

```
In [62]: print(list1)
        # Access the first element of list1
        list1[0]
```

```
[1, 2, 3]
```

```
Out[62]: 1
```

```
In [64]: print(tuple2)
        # Access the second element of tuple2
        tuple2[1]
```

```
(1, 'g', 4.4, True)
```

```
Out[64]: 'g'
```

Why do we need two very similar structures such as tuples and lists?