notebook 4. Python basics: Numbers and lists

Numeric Types — int, float, complex

There are three distinct numeric types in Python: integers (`int`), floating point numbers (`float`), and complex numbers (`complex`). In addition, Booleans (`bool`) are a subtype of integers.

`int` (integer; a whole number with no decimal place) numbers have unlimited precision.

In [ ]:
```
# example_1 for integer numbers
0, 1, 2, -1,-2000, 99999999999999999999999999999999
```

In [ ]:
```
# example_2
my_int = 6
print('value:', my_int, ', type: ', type(my_int))
```

`float` (float; a number that has a decimal place) numbers are usually implemented using double in C; information about the precision and internal representation of floating point numbers for the machine on which your program is running is available in sys.float_info.

In [ ]:
```
# example_3 for float numbers
3.14, 6.62607015*10**-34
```

In [ ]:
```
# example_4 on converting numbef from integet to float
my_float = float(my_int)
print('value:', my_float, ', type: ', type(my_float))
```

In [ ]:
```
# example_5 for precision and internal representation of floating point numbers
import sys
sys.float_info
```

`Complex` numbers have a real and imaginary part, which are each a floating point number. To extract these parts from a complex number z, use z.real and z.imag. (The standard library includes additional numeric types, fractions that hold rationals, and decimal that hold floating-point numbers with user-definable precision.)

In [ ]:
```
# example_6 for complex numbers
1+2j
```

In [ ]:
```
# example_7 for extracting real and imaginary parts from complex numbers
my_complex = 3 - 4.5j
print('value:', my_complex, ', type: ', type(my_complex))
print('value:', my_complex.real, ', type: ', type(my_complex.real))
print('value:', my_complex.imag, ', type: ', type(my_complex.imag))
```

`bool` (boolean; a binary value that is either true or false) True False

In [ ]:
```
# example_6 for boolean type
True, False
```

In [ ]:
```
my_bool = True
print('value:', my_bool, ', type: ', type(my_bool))
```

Basic operators

In Python, there are different types of **operators** (special symbols) that operate on different values. Some of the basic operators include:

- arithmetic operators
  - `+` (addition)
  - `-` (subtraction)
  - `*` (multiplication)
  - `/` (division)
  - `//` (floor division)
  - `%` (modulus division)
  - `**` (exponent)
- assignment operators
  - `=` (assign a value)
  - `+=` (add and re-assign; increment)
  - `-=` (subtract and re-assign; decrement)
  - `*=` (multiply and re-assign)
- comparison operators (return either `True` or `False`)
  - `==` (equal to)
  - `!=` (not equal to)
  - `<` (less than)

    - **<=** (less than or equal to)
    - **>** (greater than)
    - **>=** (greater than or equal to)

When multiple operators are used in a single expression, **operator precedence** determines which parts of the expression are evaluated in which order. Operators with higher precedence are evaluated first (like PEMDAS in math). Operators with the same precedence are evaluated from left to right.

- `()` parentheses, for grouping
- `**` exponent
- `*` , `/` multiplication and division
- `+` , `-` addition and subtraction
- `==` , `!=` , `<` , `<=` , `>` , `>=` comparisons

Exercises

how will look the following mathematical formulas in Python?

$$1) a = 3, b = 4, y = \sqrt{a^2 + b^2}$$

$$2) pi = 3.14, r = 5, y = 2\pi r^2$$

$$3) a = 4, b = 3, c = 2 y = 6a^3 - \frac{8b^2}{4c} + 11$$

$$4) a1 = 1, an = 10, n = 10, y = n\left(\frac{a_1 + a_n}{2}\right)$$

Python arrays

There are four collection data types in the Python programming language:

- `List` is a collection which is ordered and changeable. Allows duplicate members.
- `Tuple` is a collection which is ordered and unchangeable. Allows duplicate members.
- `Set` is a collection which is unordered and unindexed. No duplicate members.
- `Dictionary` is a collection which is unordered, changeable and indexed. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

##List A list is a collection which is ordered and changeable.

Lists may be constructed in several ways:

- Using a pair of square brackets to denote the empty list: `[]`
- Using square brackets, separating items with commas: `[a]` , `[a, b, c]`
- Using a list comprehension: `[x for x in iterable]`
- Using the type constructor: `list()` or `list(iterable)`

List items can be accessed by referring to the index number: `a[i]`

List items also can be accessed using a range of indexes by specifying where to start and where to end the range: `a[2:4]`

When specifying a range, the return value will be a new list with the specified items.

To determine if a specified item is present in a list use the `in` keyword: `1 in [1,2,3]`

note! indices starts from 0

methods of Lists :

- `.append()` add element to the end of the list
- `.remove()` to remove first specified element from the list
- `.extend()` add elements of another list to the end of the list
- `.index()` find the first index of element in the list
- `.count()` count occurrences of element in a list

functions on list:

- `len()` return lenght of a list
- `max()` return largest element
- `min()` return smallest element
- `sum()` return sum of elements if they all numbers
- `sorted()` returns sorted copy of a list

In [ ]:
```python
week = ['s', 'm', 't', 'w', 't', 'f', 's']
```