Data types, variables, operators, and input-output (IO)

Data types

Every programming language needs to store data. Each kind of data has a specific type which defines how it is stored in memory and what kind of operations that can be performed on it.

The basic data types of Python are:

- Integer: whole numbers
- Float: decimal numbers
- Complex: complex numbers
- Boolean: True/False values
- String: text
- None type: no value

Integer

In [1]:
```
5
```

Out[1]: 5

In [2]:
```
type(5)    # the type() function returns the data type of an object
```

Out[2]: int

In [3]:
```
5/2
```

Out[3]: 2.5

In [4]:
```
5//2
```

Out[4]: 2

Float

In [5]:
```
12.1
```

Out[5]: 12.1

In [6]:
```
type(12.1)
```

Out[6]: float

In [7]:
```
type(12.0)
```

Out[7]: float

In [8]:
```
round(12.1)
```

Out[8]: 12

In [9]:
```
type(round(12.1))
```

Out[9]: int

Complex

In [10]:
```
1 + 2j
```

Out[10]: (1+2j)

In [11]:
```
type(1 + 2j)
```

Out[11]: complex

String

In [12]:
```
"Hellow, world!"
```

Out[12]: 'Hellow, world!'

In [13]:
```
print("Hellow, world!")
```

```
Hellow, world!
```

Boolean

In [14]:
```
True
```

Out[14]: True

In [15]:
```
type(True)
```

Out[15]: bool

In [16]:
```
False
```

Out[16]: False

Variables

Values/objects are stored as named variables. In Python, `=` is the assignment operator. For example, `val1 = 5` is defining the variable `val1` and storing the integer `5` in it.

In [17]:
```
val1 = 5
print(val1)
```

```
5
```

Variable naming rules

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python keywords

Typecasting

Python is a dynimcally typed language, the data type of a variable (not the data itself) can change during the runtime depending on operations and assignments performed on that variable.

However, one can explicitly cast an already-defined variable to some other type.

In [18]:
```
type(val1)
```

Out[18]:
```
int
```

In [19]:
```
val1f = float(val1)
type(val1f)
```

Out[19]:
```
float
```

In [20]:
```
type(int(val1f))
```

Out[20]:
```
int
```

In [21]:
```
str(val1)
```

Out[21]:
```
'5'
```

In [22]:
```
str(val1f)
```

Out[22]:
```
'5.0'
```

In [23]:
```
bool(val1)
```

Out[23]:
```
True
```

In [24]:
```
bool(0)    # zero is coerced to false
```

Out[24]:
```
False
```

In [25]:
```
bool(-1)    # any non-zero value is considered true
```

Out[25]:
```
True
```

In [26]:
```
bool("")    # empty string is false-y
```

Out[26]:
```
False
```

In [27]:
```
bool("whatever")    # any nonempty string is truth-y
```

Out[27]:
```
True
```

In [28]:
```
str2 = "4"
```

In [29]:
```
# str2 + 5    # this will throw an error: TypeError: can only concatenate str (n
```

In [30]:
```
int(str2) + 5
```

Out[30]:
```
9
```

Operators

Operators operate on one or more objects or values to produce results. There are several kinds of operators in Python:

- Arithmetic operators `+` , `-` , `*` , `/` , `//` , `**` , `%`
- Assignment operators, `=` , `+=` , `-=` , `*=` etc
- Comparison operators, `<` , `<=` , `>=` , `==` , `!=`
- Identity operators, `is` , `is not`
- Membership operators, `in` , `not in`
- Logical operators, `and` , `or` , `not`
- Bitwise operators

Arithmetic Operaors

In [31]:
```
val1 = 5
val2 = 3
```

In [32]:
```
print(val1 + val2)
print(val1 - val2)
print(val1 * val2)
print(val1 / val2)
print(val1 // val2)    # integer division
print(val1 % val2)    # modulo operator, returns the remainder after dividing va
print(val1 ** val2)    # exponentiation
```

```
8
2
15
1.6666666666666667
1
2
125
```

Assignment Operator

- `x += 1` is equivalent to `x = x + 1`
- `x -= 1` is equivalent to `x = x -1`

In [33]:
```
val3 = 5
```

In [34]:
```
val3 += 7
val3
```

Out[34]: 12

Comparison Operator

In [35]:
```
print(val1, val2)
```

```
5 3
```

In [36]:
```
val1 == val2
```

Out[36]: False

In [37]:
```
val1 != val2
```

Out[37]: True

In [38]:
```
val1 < val2
```

Out[38]: False

In [39]:
```
val1 > val2
```

Out[39]: True

In [40]:
```
val1 >= 5
```

Out[40]: True

In [41]:
```
val1 <= 5
```

Out[41]: True

Membership Operators

- `in`
- `not in`

Checks whether an object is present inside a Python data structure (such as `list`, `dict`, `tuple`, `set` etc)

In [42]:
```
list1 = [1, 2, 3, "abc", 5.0]
list1
type(list1)
```

Out[42]: list

In [43]:
```
1 in list1
```

Out[43]: True

In [44]:
```
dict1 = {"a" : 5, "b" : 6}
"a" in dict1
```

Out[44]: True

In [45]:
```
5 in dict1
```

Out[45]: False

In [46]:
```
5 in dict1.values()
```

Out[46]: True

Logical operators

These operators perform Boolean algbera on Boolean values (`True` , `False`) that usually result from comparison, identity, or membership operators in a Python program.

In [47]:
```
val1 > val2
```

Out[47]: True

In [48]:
```
val2 < val3
```

Out[48]: True

In [49]:
```
(val1 > val2) and (val2 < val3)
```

Out[49]: True

In [50]:
```
(val1 > val2) or (val2 < val3)
```

Out[50]: True

In [51]:
```
not(val2 < val3)
```

Out[51]: False

Output (`print()` function)

The print function prints literal strings and formatted variables.

In [52]:
```
print("Hellow world!")
```

```
Hellow world!
```

In [53]:
```
print("The first value is", val1, "and the second value is", val2)
```

```
The first value is 5 and the second value is 3
```

Input (input() function)

In [54]:
```
name = input("Please enter your name: ")
```

In [55]:
```
print("Hello,", name)
```

```
Hello, Edzio
```

In [56]:
```
type(name)
```

Out[56]:
```
str
```