1_variables

October 9, 2024

1 Variables and types

File as PDF

1.1 Symbol names

Variable names in Python can contain alphanumerical characters `a-z`, `A-Z`, `0-9` and some special characters such as `_`. Normal variable names must start with a letter.

By convention, variable names start with a lower-case letter, and Class names start with a capital letter.

Note: Reserved keywords

There are a number of Python keywords that cannot be used as variable names. These keywords are:

`and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield`

Note: Be aware of the keyword `lambda`, which could easily be a natural variable name in a scientific program. But being a keyword, it cannot be used as a variable name.

1.2 Variable Assignment

The assignment operator in Python is `=`. Python is a dynamically typed language, so we do not need to specify the type of a variable when we create one.

Assigning a value to a new variable creates the variable:

[11]:
```
  # variable assignments
x = 1.0
my_favorite_variable = 12.2
x
```

[11]:
```
1.0
```

Although not explicitly specified, a variable does have a type associated with it. The type is derived from the value that was assigned to it.

[12]:
```
type(x)
```

```
[12]: float
```

If we assign a new value to a variable, its type can change.

```
[13]: x = 1
```

```
[14]: type(x)
```

```
[14]: int
```

If we try to use a variable that has not yet been defined we get an `NameError`:

```
[15]: print(g)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[15], line 1
----> 1 print(g)

NameError: name 'g' is not defined
```

1.3 Number types

Python allows as any programming language different types of variables. The type of a variable can be always accessed with the help of the *type()* command. The variable can store different data types. A list of different data types can be found in the notebook on data types. Here we treat numbers or the numeric data type a bit special.

1.3.1 Comparison of Number Types

Type	Example	Description	Limits	Use Cases
int	42	Whole numbers	Unlimited precision (bounded by available memory)	Counting, indexing
float	3.14159	Decimal numbers	Typically ±1.8e308 with 15-17 digits of precision (64-bit)	Scientific calculations, prices
complex	2 + 3j	Numbers with real and imaginary parts	Same as float for both real and imaginary parts	Signal processing, electrical engineering
bool	True / False	Logical values	Only two values: True (1) and False (0)	Conditional operations, flags

1.3.2 Integers

Python treats numbers without a decimal point automatically as an integer. In Python 2 there has been a maximum integer possible, which does not exist in Python 3 anymore.

```
[17]: # integer number
      x = 1
      type(x)
```

```
[17]: int
```

a binary number

```
[16]: 0b1010111110
```

```
[16]: 702
```

a hexadecimal number

```
[18]: 0x0F
```

```
[18]: 15
```

1.3.3 Floating Point

Floating point values are values with a decimal point. There is a maximum float in Python 3, which is 1.7976931348623157e+308. If you multiply this maximum by 2 for example, Python will threat the number as *infinity*.

```
[20]: # float variable
      x= 3.141
```

```
[21]: type(x)
```

```
[21]: float
```

```
[22]: 1.7976931348623157e+308*2
```

```
[22]: inf
```

1.3.4 Complex Numbers

```
[30]: j=5
      c=2+4j
```

```
[31]: type(c)
```

```
[31]: complex
```

```
[36]: c.imag
```

```
[36]: 4.0
```

Complex numbers have built in *accessors*. These accessors give for example access to the *real* and *imaginary* part of the complex number.

[34]:
```
r=c.real
print(r)
```

```
2.0
```

[77]:
```
i=c.imag
print(i)
```

```
4.0
```

On the other side, one my also evaluate the complex conjugate of a complex number by one of those accessors. Note that this is provided by a function here, while the above real and imaginary part are values. The are some basic functions available, which act on complex numbers. More complex calculations are possible with functions built in to modules such as *cmath* or *numpy*.

[37]:
```
c=c.conjugate()
print(c)
```

```
(2-4j)
```

1.3.5 Type conversion

Different number types can be converted into each other in python. There is either an

1) implicit
2) explicit

way of type conversion.

The **implicit conversion** can be very useful but also dangerous. Python automatically may convers one data type to another, which is known as implicit type conversion.

[38]:
```
integer_number = 123
float_number = 1.23

type(integer_number),type(float_number)
```

[38]:
```
(int, float)
```

[39]:
```
new_number=integer_number+float_number
```

[40]:
```
type(new_number)
```

[40]:
```
float
```

In **explicit type conversion**, users convert the data type of an object to required data type using the built-in functions like int(), float(), str(), etc to perform explicit type conversion.

[42]:
```
num_string = "12"
num_integer = 23
```

[43]:
```
type(num_string)
```

[43]:
```
str
```

[44]:
```
num_string = int(num_string)
```

[45]:
```
type(num_string)
```

[45]:
```
int
```

[46]:
```
x = 5/2

print(x, type(x))
```

```
2.5 <class 'float'>
```

[47]:
```
x = int(x)

print(x, type(x))
```

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

[111]:
```
z = complex(x)

print(z, type(z))
```

```
(2+0j) <class 'complex'>
```

[97]:
```
type(z.imag)
```

[97]:
```
float
```

Complex variables cannot be cast to floats or integers. We need to use `z.real` or `z.imag` to extract the part of the complex number we want:

[98]:
```
y = bool(z.real)

print(z.real, " -> ", y, type(y))

y = bool(z.imag)

print(z.imag, " -> ", y, type(y))
```

```
2.0  ->  True <class 'bool'>
0.0  ->  False <class 'bool'>
```