# 1_variables

April 12, 2024

## 1 Variables and types

### 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:

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

```
[3]: 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.

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

```
[2]: float
```

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

```
[3]: x = 1
```

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

```
[4]: int
```

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

```
[5]: print(y)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-d9183e048de3> in <module>
----> 1 print(y)

NameError: name 'y' 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.

### 1.3.1 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.

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

```
[6]: int
```

### 1.3.2 Floating Point

Floating point values are values with a decimal point. Also here a maximum float is non existant in Python 3.

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

### 1.3.3 Complex Numbers

```
[8]: c=2+4j
     type(c)
```

```
[8]: complex
```

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

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

```
2.0
```

```
[10]: 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*.

```
[11]: c=(2+4j).conjugate()
      print(c.imag)
```

```
-4.0
```

### 1.3.4 Type casting

```
[4]: x = 1.5

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

```
1.5 <class 'float'>
```

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

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

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

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

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

```
(1+0j) <class 'complex'>
```

```
[7]: x = float(z)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-19c840f40bd8> in <module>
----> 1 x = float(z)

TypeError: can't convert complex to 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:

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

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

y = bool(z.imag)

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

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