In this lecture we learn more about functions and their input

We emphasize and understand the difference:

- between function *arguments* and function *parameters*
- between function *definition* and function *evaluation* (call)
- between function *positional* arguments and function *keyword* arguments
Functions in Mathematics

In Mathematics: A function is written as a map, that uniquely assigns an element $y$ from the range $R$ to every element $x$ from the domain $D$.

$$f : x \mapsto y.$$ $f$ is the function
$x$ is its argument
$y$ is its (return) value

There can be several arguments of different type:
Consider as an example the function,

$$f(g, a, b) = \int_a^b g(x)dx$$

The input arguments cannot be swapped! Position matters.
Functions in Python

Definition of a function:

```python
def f(x1, x2, x3):
    .....  # some computations
    y = ...  # assign a value to y to return to user
    return y
```

Evaluation (call) of a function:

```python
f(17, 18, -2)
f([1, 2, 3], {'Tol': 1.e-10}, 'ro')
...
```

Wording:

x1, x2, x3 are called function parameters (needed for the definition)
17, 18, -2 are called arguments (needed for the evaluation)
Passing Arguments - by position or by keyword

Consider the following example:

```python
def subtract(x1, x2):
    return x1 - x2
```

Typical use: passing arguments by position:

```python
subtract(1, 2)  # returns -1
```

Position matters.

Another way: passing arguments by keyword:

```python
subtract(x2=2, x1=1)  # returns -1
```

Position doesn’t matter.
Scope of variables I

Variables defined inside the function are said to belong to the function’s **scope**.
They are unknown outside the function.

**Definition:**

```python
def mult2(x):
    c = 2.
    return c*x
```

**Evaluation:**

```python
mult2(20.)  # returns 40
c            # returns NameError: 'c'
    not defined
```
Scope of variables II

Compare:

**Example 1.**
a is a parameter of the function

```python
def multiply(x, a):
    return a*x
```

**Example 2.**
a is referenced from outside the function’s scope:

```python
a = 3
def multiply(x):
    return a*x
```

We encourage programming practices which resemble Example 1 rather than Example 2.
Local variables - can they be changed?

As a principle, never change input variables within a function. **Immutable** variables are copied, not changed, when calling a function.

Example: you cannot change an input variable, \( a \), by changing the local variable \( x \):

```python
a = 3          # a is 3
def f(x):
    x = 2      # changed the input x to 2
    return x
print(f(a))   # result from function is 2
print(a)      # a is still 3
```

*Be careful! You can change mutable variables like lists, dictionaries with the example above!!!*
Mandatory and Default Arguments

default (eng) = standard (sv)

The standard way ...

\[ x, c \text{ are mandatory arguments} \]

\[
\text{def mult}(x, c):
    \text{return } c \times x
\]

a typical call

\[
y = \text{mult}(2., 13.)
# returns 26
\]
\[
y = \text{mult}(c=13., x=2.)
# returns 26
\]

... now with \( x \) as mandatory and \( c \) as default argument

\[
\text{def mult}(x, c=1.0):
    \text{return } c \times x
\]

possible calls

\[
y = \text{mult}(2., 13.)
# returns 26
\]
\[
y = \text{mult}(2.)
# returns 2
\]

In the definition of the function mandatory parameters must precede optional parameters (those with default values). Why?
Functions, Function Arguments, Function Parameters – Summary

Make sure that you understood the difference:

- between function **arguments** and function **parameters**
- between function **definition** and function **evaluation** (call)
- between function **positional** arguments and function **keyword** arguments
Docstrings

All functions (and everything else) should be documented carefully:

A docstring is the leading comment in a function (or class):

```python
def newton(f, x0):
    """
    Newton's method for computing a zero of a function
    on input:
    f (function) given function f(x)
    x0 (float) initial guess
    on return:
    y (float) the approximated zero of f
    """
    ...
```

`help(newton)` in Python or `newton?` in IPython displays the docstring as a help text.