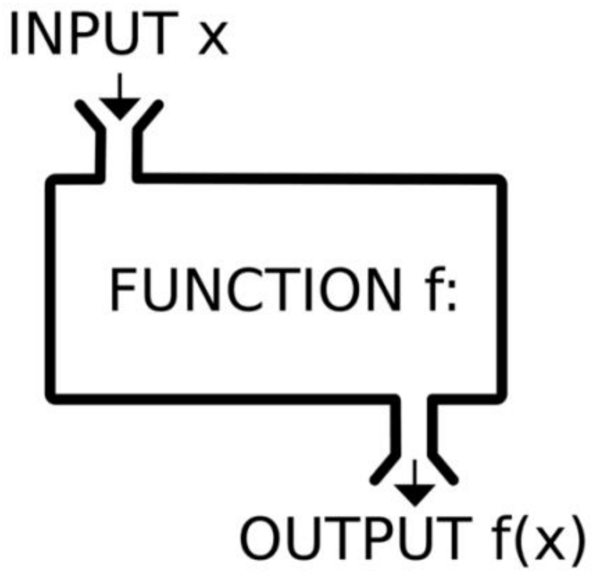
COMP 125 Programming with Python
Back to Functions



Mehmet Sayar

Koç University

Midterm 3

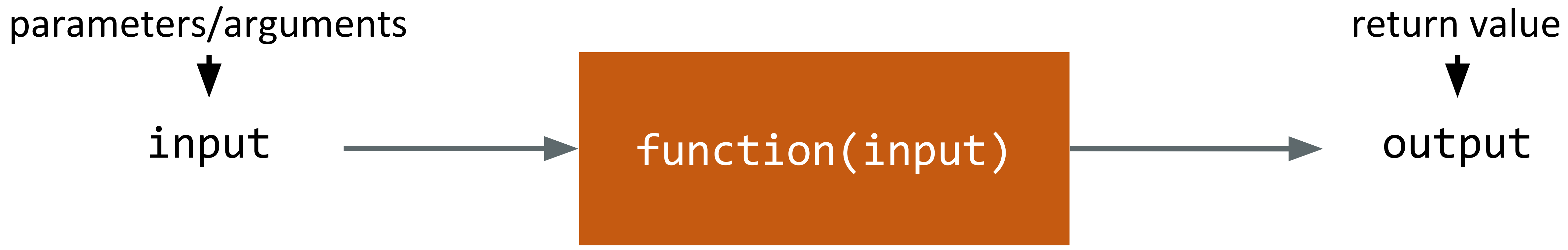
Sunday Dec. 6

8:30-11:30

Homework 3

Dec 8 by midnight

Remember: Functions



Definitions

parameter(s)
One or more variables that your function expects as input

argument(s)
The values passed into your function and assigned to its parameter variables

Definition

return value
The value that your function hands back to the “calling” function

Some Questions – Do we know their answer?

- Multiple Inputs? } Yes
- Multiple Outputs? } Sort of
- Default Arguments?
- Do we have to remember input order?

} No

- Importing modules that have functions? } Yes
- Importing only specific functions from a module? } No

Note: all the answers are "yes, it is possible"

Multiple Inputs

- List them in the function header!

```
def func(param1, param2, param3 ...)
    do something with the params
```

- Any questions?
- I have one, what if we need “variable number of inputs”? (e.g. `print`)
    - Later in the course if we have time left 😊
    - For the curious: the * and ** operators for unpacking and the *args, **kwargs statements

Remember: Packing and Unpacking

- Packing: Assigning multiple values (or an iterable) to a tuple
  - The no parentheses example

```
tup = 'elma', 7.99, 'market'
```

  - Possible to do packing with a list. We will see more later on

- Unpacking: Assigning the contents of the tuple to multiple values

```
urun, fiyat, satici = tup
urun → 'elma'
```

  - Other iterables can also be unpacked!

Multiple Outputs

- Just return a collection (list, tuple, dictionary etc.)!
- Default behavior that returns a tuple:

```
def func(…)
    ...
    return ret_val1, ret_val2, …
a,b, … = func(…)
```

- May also put brackets around the return values to return a list or parentheses to emphasize the tuple behaviour (both unpack the same)
- Any questions?

Default Arguments

- What if we do not want to specify all the parameters of a function?
- Example: Remember the `split` method of strings?

```
sentence = "the apples, the oranges, and the coconut?"
sentence.split()
→ ['the', 'apples,', 'the', 'oranges,', 'and', 'the',
'coconut?']
```

Default Arguments

```
sentence = "the apples, the oranges, and the coconut?"
sentence.split(',')
→ ['the apples', ' the oranges', ' and the coconut?']
```

Pay attention to spaces!

```
sentence.split(', ')
→ ['the apples', 'the oranges', 'and the coconut?']

sentence.split('the')
→ ['', ' apples, ', ' oranges, and ', ' coconut?']
```

Default Arguments

- The most common split is done with white spaces which are the default values
- You can do the same with other functions as follows:

```
def func(param1 = def_arg1, param2 = def_arg2, ...):
    do something with the params
```

- The order stays left to right
- Unspecified arguments (based on this order) is taken as the default value.

Default Arguments

```
def calculate_box_volume(length = 1.0, width = 1.0, height = 1.0):
    return length*width*height
```

Outputs?

```
calculate_box_volume()
calculate_box_volume(2)
calculate_box_volume(2,3)
calculate_box_volume(2,3,4)
```

Default Arguments

- Can specify default or only to a subset of parameters as well:

```
def func(param1, param2 = def_arg2, ...):
    do something with the params
```

- Example

```
def calculate_box_volume(length, width = 1.0, height = 1.0):
    return length*width*height
```

Outputs?

```
calculate_box_volume() (error!)
calculate_box_volume(2)
calculate_box_volume(2,3)
calculate_box_volume(2,3,4)
```

Default Arguments

- The parameters with default arguments must come later than the parameters without in the function header:

```
def func(param1 = def_arg1, param2i, ...):
    do something with the params
SyntaxError: non-default argument follows default argument
```

Keyword Arguments

- What to do if we do not remember the order or specify inputs by their name?

```
def calculate_box_volume(length = 1.0, width = 1.0, height=1.0):
    return length*width*height
```

- Call them by their name!
- The below uses the default for `length` and `height` but the given value for `width`

```
calculate_box_volume(width = 2.5)
```

Keyword Arguments (More on Spyder)

```
def calculate_box_volume(length = 1.0, width = 1.0, height=1.0):
     return length*width*height
```

- You can’t call positional arguments after the keyword arguments
  - `calculate_box_volume(1, width = 2.5) (length=1.0): Legal`
  - `calculate_box_volume(width = 2.5,1) (ambigious): Illegal`

- The order of keyword arguments does not matter
  - `calculate_box_volume(width = 2.5, length=0.5)`

How to transfer information to functions?

- Pass-by-value
  - Copies the value and passes the copy
- Pass-by-reference
  - Allows a function to access the caller data and to modify it
  - Can increase performance
    - Prevents the copying of large data
  - Can weaken security
    - Allows direct access to the data
- Pass by object reference
  - Only thing allowed in Python
  - Combination of pass-by-value and pass-by-reference
  - Can modify mutable objects and not immutable objects

Passing Lists to Functions

- Passing a list
    - The same applies for other mutable objects in Python
    - To pass a list pass it without its brackets
    - This allows the entire list to be changed
    - Item in the list that are immutable (numbers or strings) cannot be changed by the function when passed individually

Spyder demo