Programing in Python

Lecture 4 - Functions

Instructor: Zhandos Yessenbayev

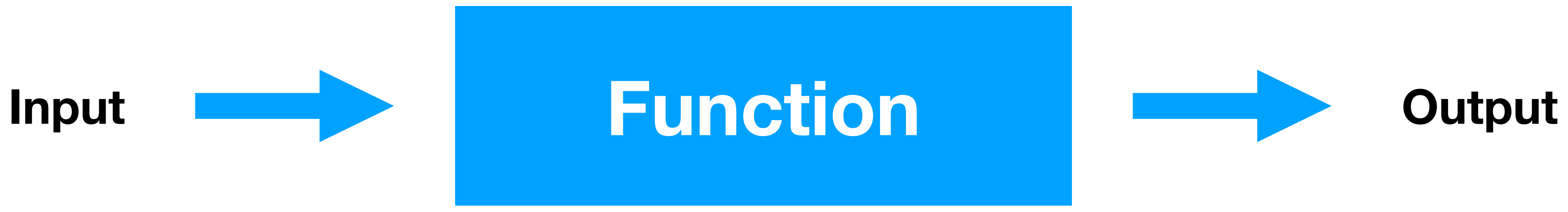
Outline

- What is a function?
- Built-in Functions
- User-defined functions
- Function parameters
- Returning a result

What is a function?

- **Function** is a *named* sequence of statements that:
  - takes some *input* as **parameters**
  - *performs* a **computation**
  - *outputs* a **result**

Examples: print('Hello'), input( ), int('25'), str(25)



What is a function?

- There are **two** types of functions in Python:
  - Built-in functions
  - User-defined functions

Built-in Functions

- There are about 70 built-in functions in Python:
    - print(), input()
    - int(), float, str()
    - len(), min(), abs(), …
    - Some mathematical and system functions

More functions: https://docs.python.org/3.8/library/functions.html

Built-in Functions

- We can easily write programs with built-in functions:

```
>>> n = int(input("Enter a number: "))
Enter a number: 10
>>> x = 2 * n + 1
>>> print("New number: ", float(x))
New number:  21.0
>>>
```

User-defined functions

- Working with functions has 2 stages:

  1. Function **definition** (store in memory)

  2. Function **execution** (reuse in program)

How does it work? 🤔

Function definition

- We **define** a function with the keyword **def**
- General syntax:

  def <function_name> (<function_parameters>) :
  <function_body>

- Notice also colon and indentation of function body

```
>>> def myfunc():
...     print("This is my function")
...     print("I can only print this text")
...
>>>   ←──          nothing is printed here!!!
```

Function execution

- To **execute** the defined function, just call it by name with brackets:

Compare these!!!

```
>>> myfunc()
This is my function
I can only print this text
>>>
```

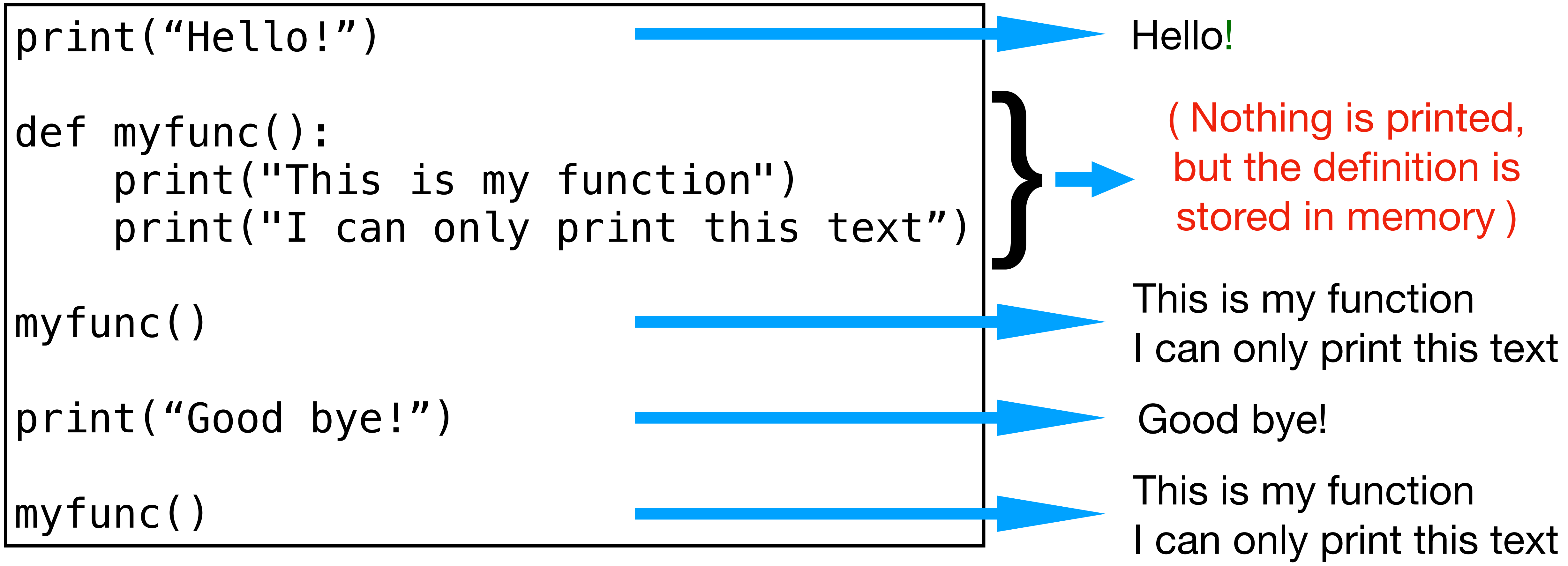
```
>>> myfunc
<function myfunc at 0x108f76af0>
```

- In script mode:

```
def myfunc():
    print("This is my function")
    print("I can only print this text”)

myfunc()
```

Flow of program

- Let’s see what happens in this program



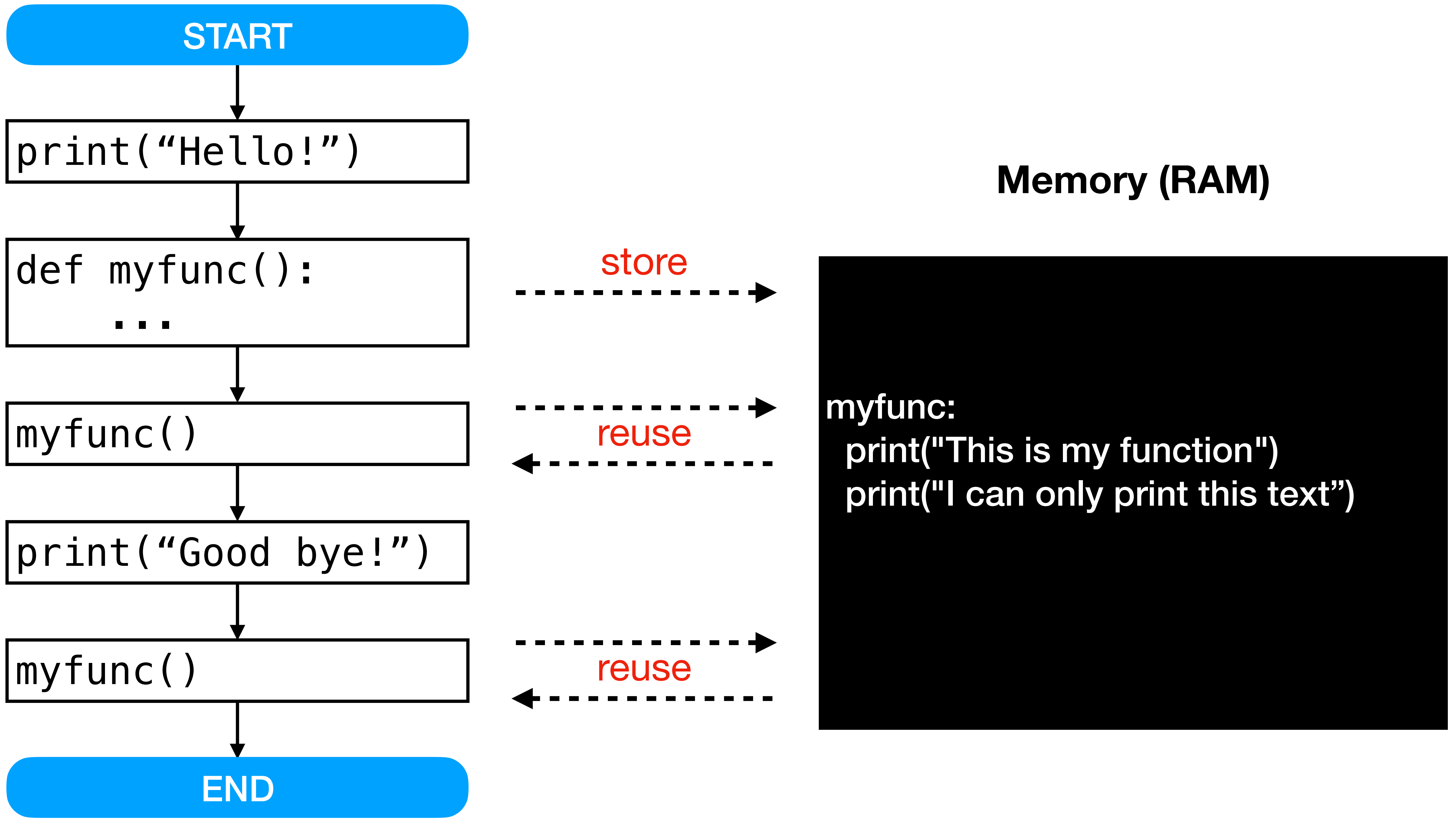
Program

```
print(“Hello!”)

def myfunc():
    print("This is my function")
    print("I can only print this text”)

myfunc()

print(“Good bye!”)

myfunc()
```

Output

```
Hello!

( Nothing is printed,
but the definition is
stored in memory )

This is my function
I can only print this text

Good bye!

This is my function
I can only print this text
```

Flow of program



Exercise

Write a program that *defines* a function with the name **greet( ),** which prints the text “Hello, world!”, and *executes* it 5 times using **for**-loop.

Exercise (solution)

```
def greet():
    print("Hello, world!")

for n in [1,2,3,4,5]:
    greet()
```

Function parameters

- Functions can take **input** to perform computations
- Input is passed as **parameters** of the function
- Remember the general syntax:

  def <function_name> (**<function_parameters>**) :

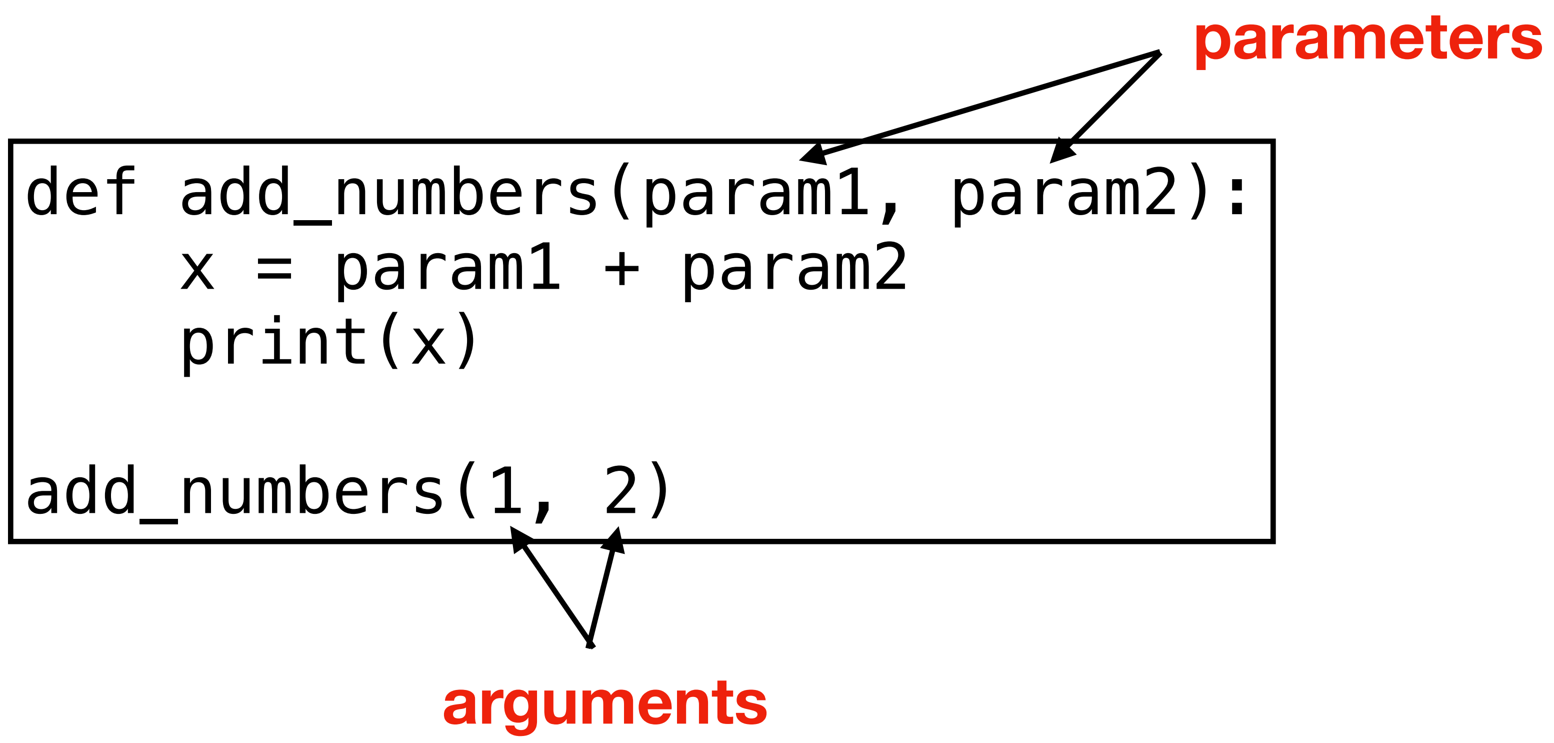
    <function_body>

```
>>> def print_myname(name, last_name):
...     print("My name is", name, last_name)
...
>>> print_myname(“Zhandos”, “Yessenbayev”)
My name is Zhandos Yessenbayev
>>>
```

Function parameters

- Function **parameters** are the ***names (variables)*** listed in the function's definition.

- Function **arguments** are the real ***values*** passed to the function.



```
def add_numbers(param1, param2):
    x = param1 + param2
    print(x)

add_numbers(1, 2)
```

Function parameters

- Names of the **variables** and **parameters** are local, i.e. seen only inside the function

These are different variables

```
x = 5

def change_x(param):
    x = param * 2

change_x(1)
print(x)

change_x(x)
print(param)
```

???

Returning a result

- Sometimes functions may *return* the results of the computations.
- To return the results, we use the keyword **return**

```
def add_numbers(param1, param2):
    x = param1 + param2
    return x

a = add_numbers(1, 2)
print(a)
```

Returning a result

- Everything ***after*** the **return** word is ignored
- To return the results, we use the keyword **return**

```
def divide(param):
    x = param / 2
    return x
    x = param * 2

a = divide(10)
print(a)
```

ignored

Arbitrary Arguments

- If we don’t know how many arguments will be passed to the function, we can use * symbol before the parameter:

```
>>> def f(*args):
...   for i in args:
...     print(i)
...
>>> f(1,2,3,4,5,6,7)
>>> f(2,4,6)
```

Passing List as Argument

- We can pass the whole list as arguments

```
>>> def g(lst):
...   for i in lst:
...     print(i)
...
>>> g([1,2,3,4,5,6,7])
>>> g([2,4,6])
```

Keyword Argument

- We can give arguments with key=value syntax
- In this case, the order of arguments does not matter

```
>>> def greet(name, last_name):
...   print("Hello, ", name, last_name)
...
>>> greet("jan", "yessen")
Hello,  jan yessen
>>> greet("yessen", "jan")
Hello,  yessen jan
>>> greet(last_name="yessen", name="jan")
Hello,  jan yessen
```

Default Parameter Value

- Sometimes we can provide default value for the parameters.
- If we call the function without parameters, it will use default values for the parameters&

```
>>> def greet(name=“Tom”, last_name=“Cruz”):
...   print("Hello, ", name, last_name)
...
>>> greet("jan", "yessen")
Hello,  jan yessen
>>> greet()
Hello,  Tom Cruz
```

The pass Statement

- The body of the function cannot be empty, but it can do nothing with pass statement

- The minimal function is:

```
>>> def my_func():
...    pass
...
>>> my_func()
```

Recursion

- Recursion is a technique where a function can call itself
- It is important to write stopping condition!!!

```
>>> def count_down(n):
...   # stopping condition
...   if n == 0:
...     return
...
...   # do some computation
...   print(n)
...
...   # call itself
...   count_down(n-1)
...
>>> count_down(10)
```

Thanks!