Maktab Sharif

Python | Main course

Session 3

Range, else in loops
Functions
Built-in functions
Local and Global variables
(Python Short-Hands)

by Mohammad Amin H.B. Tehrani - Reza Yazdani

www.maktabsharif.ir

Review

String formatting

Maktab Sharif

formats the specified value(s) and insert them inside the string's placeholder.

- ➔ **F-strings (python +3.6):** exp: `f"hello {name}"`
- ➔ **.format() string method (new):** exp: `"hello {}".format(name)`
- ➔ **% operator (old):** exp: `"hello %s" % name`

```
name = 'Akbar'
age = 21

print("Hello "+name+' (Age:'+str(age)+')')
print(f"Hello {name} (Age:{age})")
print("Hello {} (Age:{})".format(name, age))
print("Hello {1} (Age:{0})".format(age, name))
print("Hello {name} (Age:{age})".format(age=age, name=name))
print("Hello %s (Age:%d)" % (name, age))
print("Hello %s (Age:%d)" % (name, age))
```

range() function

Maktab Sharif

Generate a sequence of numbers: **range(...)**

- ➔ **range(stop)**: exp: range(5) -> 0, 1, 2, 3, 4
- ➔ **range(start, stop)**: exp: range(5, 10) -> 5, 6, 7, 8, 9
- ➔ **range(start, stop, step)**: exp: range(5, 10, 2) -> 5, 7, 9

```
print(type(range(1,2)))          # ?

for i in range(8):
    print(i, end='')             # end??

for i in range(1, 6, 2):
    print(i, end=', ')

numbers = list(range(0,10))      # Cast range() to list
print(numbers)                   # ???
```

Else in loops

Maktab Sharif

The else block just after for/while is executed only when the loop is NOT terminated by a break statement.

```
n = 10
_ = 0
while _ < n:
    print(_)
    if _ // 10:
        print('Breaking!')
        break
    _ += 1
else:
    print('Else part!')
print('Rest of program...')
```

Example: Is Primal

Maktab Sharif

Primal number

Write a program that, gets a positive int number,
Then prints that it's primal or not

Input 1:

```
51
```

Output 1:

```
False
```

Input 2:

```
23
```

Output 2:

```
True
```

Example: Is Primal

Maktab Sharif

Primal number

Write a program that, gets a positive int number,
Then prints that it's primal or not

```
n = int(input("Enter a number: "))

for _ in range(2, n):
    if n % _ == 0:
        print('False')
        break
else:
    print('True')
```

Chapter 9

Functions

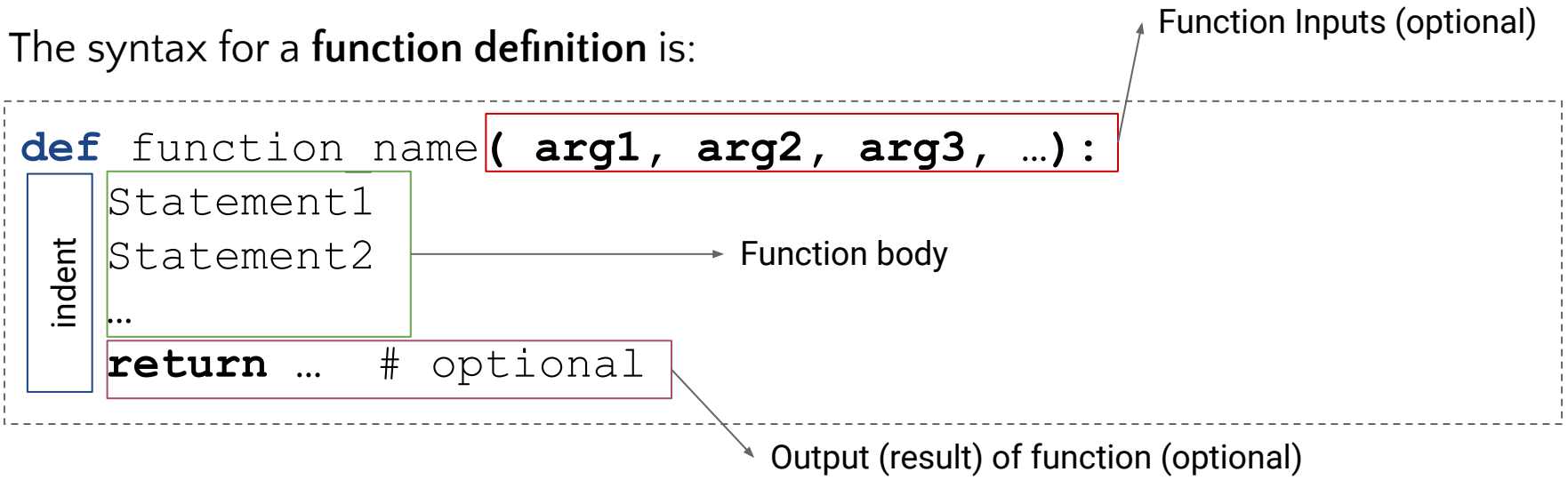
Maktab Sharif

Define a function

Maktab Sharif

In Python, a **function** is a named sequence of statements that belong together. Their primary purpose is to help us organize programs into chunks that match how we think about the solution to the problem.

The syntax for a **function definition** is:



Function return statement

A **return** statement is used to **END** the execution of the function call and "returns" the result (value of the expression following the return keyword) to the caller.
The statements after the return statements are **NOT** executed.

➔ If the return statement is without any expression, then the special value **None** is returned.

```
def function_name( arg1, arg2, arg3, …):
    Statement1
    Statement2
    …
    return …   # optional
```

Output (result) of function (optional)

Function call

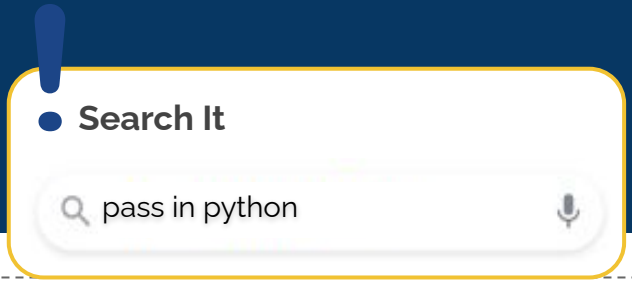
Maktab Sharif

To call a function, use the function name followed by parenthesis.

```
def function_name( arg1, arg2, arg3, …):
    Statement1
    Statement2
    …
    return …  # optional

result = function_name(1, 'akbar', True, …)
```

Function Examples



Maktab Sharif

```
def hello_world():
    print('Hello world')

res = hello_world()
print(type(res))
print(res)
```

```
def pow(a, b):
    return a ** b

res = pow(2, 4)
print(res)
print(type(res))
```

```
def my_func(arg1, arg2, arg3):
    print('> Start of function')
    print(arg1, arg2, arg3)
    return arg1, arg2, arg3
    print('> End of function')

res = my_func('Akbar', 111, False)
print(res, res[0], res[1], res[2])
print(type(res))   #???
```

```
def pass_func(x, y):
    pass

print(pass_func(1, 2))
```

Optional arguments (default value)

Maktab Sharif

Functions can have optional parameters, also called default parameters. Default parameters are parameters, which don't have to be given, if the function is called. In this case, the default values are used. We will demonstrate the operating principle of default parameters with an example. The following little script, which isn't very useful, greets a person. If no name is given, it will greet everybody:

```
def say_hello(name='Akbar'):
    print('Hello', name, '!')

say_hello()
say_hello('Reza')
say_hello('World')
```

```
def pow(base, exponent=2):
    return base ** exponent

print(pow(4))
print(pow(2, 6))
print(pow())
```

Example: Primal numbers

Maktab Sharif

Primal number

Write a **function** that, gets a positive int number (N),
Then prints primal numbers **between 1 and N.**

Input 1:

```
7
```

Output 1:

```
2, 3, 5, 7,
```

Input 2:

```
23
```

Output 2:

```
2, 3, 5, 7, 11, 13, 17, 19, 23,
```

Example: Primal numbers

Maktab Sharif

Primal number

Write a **function** that, gets a int & positive number (N),
Then prints primal numbers **between 1 and N.**

```
# Part 1: is_primal function
def is_primal(n):
    for _ in range(2, n):
        if n % _ == 0:
            return False
    else:
        return True
```

```
# Part 2: main
N = int(input('Enter a number:'))

for n in range(2, N+1):
    if is_primal(n):
        print(n, end=', ')
```

Keyword arguments (Named parameters)

Maktab Sharif

Using keyword parameters is an alternative way to make function calls. The definition of the function doesn't change.
An example:

```
def sum_sub(a, b, c=0, d=0):
    return a - b + c - d


print(sum_sub(12, 4))
print(sum_sub(b=5, a=10))
print(sum_sub(42, 15, 11, 10))
print(sum_sub(42, 15, 0, 10))
print(sum_sub(42, 15, d=10))
print(sum_sub(b=2, 5, 4))          #???
```

Local and Global variables

Maktab Sharif

Using keyword parameters is an alternative way to make function calls. The definition of the function doesn't change.
An example:

```
def square1(x):
    y = x * x
    return y

z = square1(10)
print(y)
```

```
def square2(x):
    y = x ** power
    return y

power = 2
result = square2(10)
print(result)
```

Local and Global variables

Maktab Sharif

Using keyword parameters is an alternative way to make function calls. The definition of the function doesn't change.
An example:

```
def func2():
    x *= 10

x = 10
func2()
```

```
def func1():
    print(x)

x = 10
func1()
```

What is your conclusion?

Example: Swap function

Maktab Sharif

Swap function

Write a function that, gets 2 values
Then swaps them.

Inputs:

```
def swap(...):
    ...

a = 10
b = 'Akbar'
swap()
print(a, b)
```

result:

```
Akbar 10
```

Example: Swap function

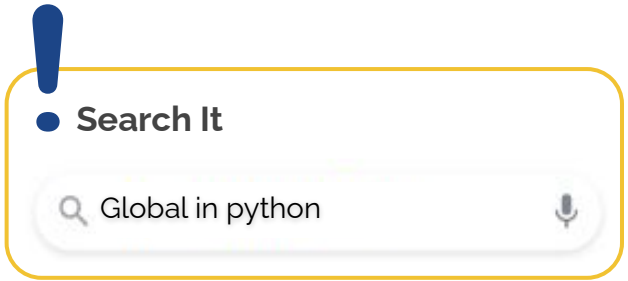
Maktab Sharif

Swap function

Write a function that, gets 2 values
Then swaps them.

Using **global**:

```
def swap():
    global a
    global b
    a, b = b, a
```



Example

Maktab Sharif

What's result of code below

```
def hi_all(students_list, teacher_list):
    teacher_list.extend(students_list)
    for x in teacher_list:
        print('> Hello', x, '!')


teachers = ['Shahin', 'Amirhossein', 'MohammadAmin']
students = ['Ali', 'Mohammad', 'Salar', 'Akbar', 'Nader']
print('Before function:', students, teachers, '', sep='\n\t')
hi_all(students, teachers)
print('\nAfter function:', students, teachers, sep='\n\t')
```

Example

Maktab Sharif

Why??

```
Before function:
     ['Ali', 'Mohammad', 'Salar', 'Akbar', 'Nader']
     ['Shahin', 'Amirhossein', 'MohammadAmin']

> Hello Shahin !
> Hello Amirhossein !
> Hello MohammadAmin !
> Hello Ali !
> Hello Mohammad !
> Hello Salar !
> Hello Akbar !
> Hello Nader !

After function:
     ['Ali', 'Mohammad', 'Salar', 'Akbar', 'Nader']
     ['Shahin', 'Amirhossein', 'MohammadAmin', 'Ali', 'Mohammad', 'Salar', 'Akbar', 'Nader']
```

Local and Global variables

Maktab Sharif

Using keyword parameters is an alternative way to make function calls. The definition of the function doesn't change.
An example:

```
def square1(x):
    y = x * x
    return y

z = square1(10)
print(y)
```

```
def square2(x):
    y = x ** power
    return y

power = 2
result = square2(10)
print(result)
```

Local and Global variables

Maktab Sharif

Using keyword parameters is an alternative way to make function calls. The definition of the function doesn't change.
An example:

```
def func2():
    x *= 10

x = 10
func2()
```

```
def func1():
    print(x)

x = 10
func1()
```

What is your conclusion?

Example: Swap function

Maktab Sharif

Swap function

Write a function that, gets 2 values
Then swaps them.

Inputs:

```
def swap(...):
    ...

a = 10
b = 'Akbar'
swap()
print(a, b)
```

result:

```
Akbar 10
```

Example: Swap function

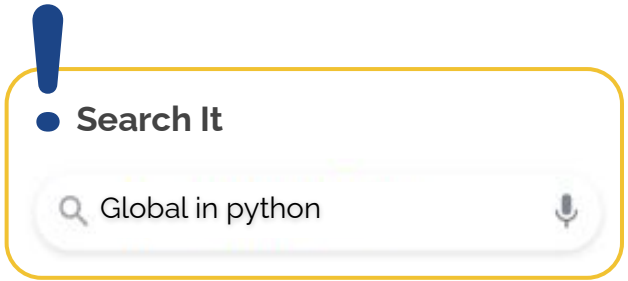
Maktab Sharif

Swap function

Write a function that, gets 2 values
Then swaps them.

Using **global**:

```
def swap():
    global a
    global b
    a, b = b, a
```



Example

Maktab Sharif

What's result of code below

```
def hi_all(students_list, teacher_list):
    teacher_list.extend(students_list)
    for x in teacher_list:
        print('> Hello', x, '!')


teachers = ['Shahin', 'Amirhossein', 'MohammadAmin']
students = ['Ali', 'Mohammad', 'Salar', 'Akbar', 'Nader']
print('Before function:', students, teachers, '', sep='\n\t')

hi_all(students, teachers)
print('\nAfter function:', students, teachers, sep='\n\t')
```

Example

Maktab Sharif

Why??

```
Before function:
     ['Ali', 'Mohammad', 'Salar', 'Akbar', 'Nader']
     ['Shahin', 'Amirhossein', 'MohammadAmin']

> Hello Shahin !
> Hello Amirhossein !
> Hello MohammadAmin !
> Hello Ali !
> Hello Mohammad !
> Hello Salar !
> Hello Akbar !
> Hello Nader !

After function:
     ['Ali', 'Mohammad', 'Salar', 'Akbar', 'Nader']
     ['Shahin', 'Amirhossein', 'MohammadAmin', 'Ali', 'Mohammad', 'Salar', 'Akbar', 'Nader']
```

Chapter 10

Some built-in function

Maktab Sharif

type(x)

Maktab Sharif

The **type()** function returns the type of the specified object

```
print(type(""))
print(type(124))
print(type(124.5))
print(type('124'))
print(type([]))
print(type(True))
print(type(range(5)))
print(type(str('123')))
```

len(...)

Maktab Sharif

The **len()** function returns the number of items in an object.

```
int_list = [1, 2, 3, 4, 5]
print(len(int_list))

float_tuple = (12.5, -2, 1.25, 0.5)
print(len(float_tuple))

print(len('Hello world!'))
string = "d e g a b l c f i h k j n"
print(len(string))

str_set = set('abcd')
print(str_set, len(str_set))
```

abs(num)

Maktab Sharif

The **abs()** function returns the absolute value of the specified number.

```
print(abs(-12))
print(abs(12))
print(abs(-0.5))
print(abs(0.5123))
print(abs(False))
print(abs('-12'))
```

round(float_num)

Maktab Sharif

The **round()** function returns a floating point number that is a rounded version of the specified number, with the specified number of decimals.

```
print(round(-12.8765, 2))
print(round(12.1245, 5))
print(round(-0.5342, 2))
print(round(0.5123, 2 ))
print(round(False, 1))
print(round('-12', 5))
```

sum(num_list)

Maktab Sharif

The **sum()** function returns a number, the sum of all items in an **list**.

```
int_list = [1,2,3,4,5]
print(sum(int_list))

float_list = [12.5, -2, 1.25, 0.5]
print(sum(float_list))

num_string = "12 5.5 -2.5 11"
num_str_list = num_string.split()
num_float_list = list(map(float, num_str_list))
print(sum(num_float_list))
```

max(num_list)

Maktab Sharif

The **max()** function returns the item with the highest value, or the item with the highest value in an iterable.

```
int_list = [1,2,3,4,5]
print(max(int_list))

float_list = [12.5, -2, 1.25, 0.5]
print(max(float_list))

num_string = "12 5.5 -2.5 11"
num_str_list = num_string.split()
num_float_list = list(map(float, num_str_list))
print(max(num_float_list))
```

min(num_list)

Maktab Sharif

The **min()** function returns the item with the lowest value, or the item with the lowest value in an iterable.

```
int_list = [1,2,3,4,5]
print(min(int_list))

float_list = [12.5, -2, 1.25, 0.5]
print(min(float_list))

num_string = "12 5.5 -2.5 11"
num_str_list = num_string.split()
num_float_list = list(map(float, num_str_list))
print(min(num_float_list))
```

sorted(list)

Maktab Sharif

The **sorted()** function returns a sorted list of the specified iterable object.

```
int_list = [1,2,3,4,5]
sorted_i_list = sorted(int_list)
print(sorted_i_list)

float_list = [12.5, -2, 1.25, 0.5]
sorted_f_list = sorted(float_list)
print(sorted_f_list)

string = "d e g a b l c f i h k j n"
str_list = string.split()
sorted_s_list = sorted(str_list)
print(sorted_s_list)
```

map(x, list)

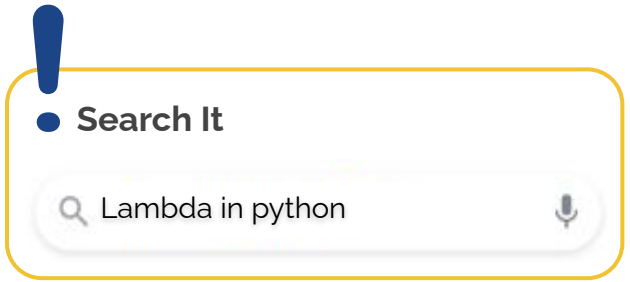
Maktab Sharif

The **map()** function executes a specified function for each item in an iterable. The item is sent to the function as a parameter.
We can use it to convert all of list items to specific type.

```
str_list = ['-1','13.5', '-2.1', '11']
print(str_list)

float_list = list(map(float, str_list))
print(float_list)

final_list = list(map(lambda x: int(x)**2, float_list))
print(final_list)
```



More built-in functions

`filter()`	`enumerate()`	`round()`	`bin()`	`sorted()`	`zip()`
`eval()`	`chr()`	`any()`	`hex()`	`reversed()`	`divmod()`
`exec()`	`ord()`	`all()`	`oct()`	`format()`	

Chapter 11

(Short-Hands)

Maktab Sharif

Conditional expression (ternary operator)

Maktab Sharif

Ternary operators also known as conditional expressions are operators that evaluate something based on a condition being true or false.

```
variable = first_value if condition else second_value
```

```
a, b = 10, 20

minimum = a if a < b else b

print(minimum)
```

```
a, b = 10, 20

if a < b:
    minimum = a
else:
    minimum = b

print(minimum)
```

List Comprehension (inline for)

Maktab Sharif

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

```
newlist = [expression for item in iterable]
newlist = [expression for item in iterable if condition == True]
```

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

List Comprehension Examples

Maktab Sharif

```
l = [x**2 for x in range(1, 21)]
print(l)
```

```
s = 'Akbar neshan'
l = [x.upper() for x in s if x.lower() in 'aoieu']
print(l)
```

```
n = 10

for i in range(1, n+1):
    print(*[i*j for j in range(1, n+1)], sep='\t')
```

Lambda (inline function)

Maktab Sharif

A lambda function is a small anonymous function.
A lambda function can take any number of arguments, but can only have one expression.

`lambda` *`arguments`* : *`expression`*

```
f = lambda x: 2*x + 10
print(f(2))
```

```
def f(x):
    return 2*x + 10

print(f(2))
```

Lambda Examples

Maktab Sharif

```
f = lambda a, b, c : a + b + c
print(f(5, 6, 2))
```

```
l = list(map(lambda x: x ** 2, range(1, 21)))
print(l)
```

```
n = int(input('Enter a number: '))

x = lambda n: not [i for i in range(2, n) if not (n % i)]
y = lambda N: [i for i in range(2, N+1) if x(i)]

print(y(n))
```

Pre-reading

Search about:

1. List Comprehensions (inline-for) in python
2. Conditional expression (ternary operator or inline-if) in python
3. * Tuple in python (Tuple vs. list)
4. * Set in python (Set vs. list)
5. Lambda functions in python
6. args in python (*args)
7. kwargs in python (**kwargs)