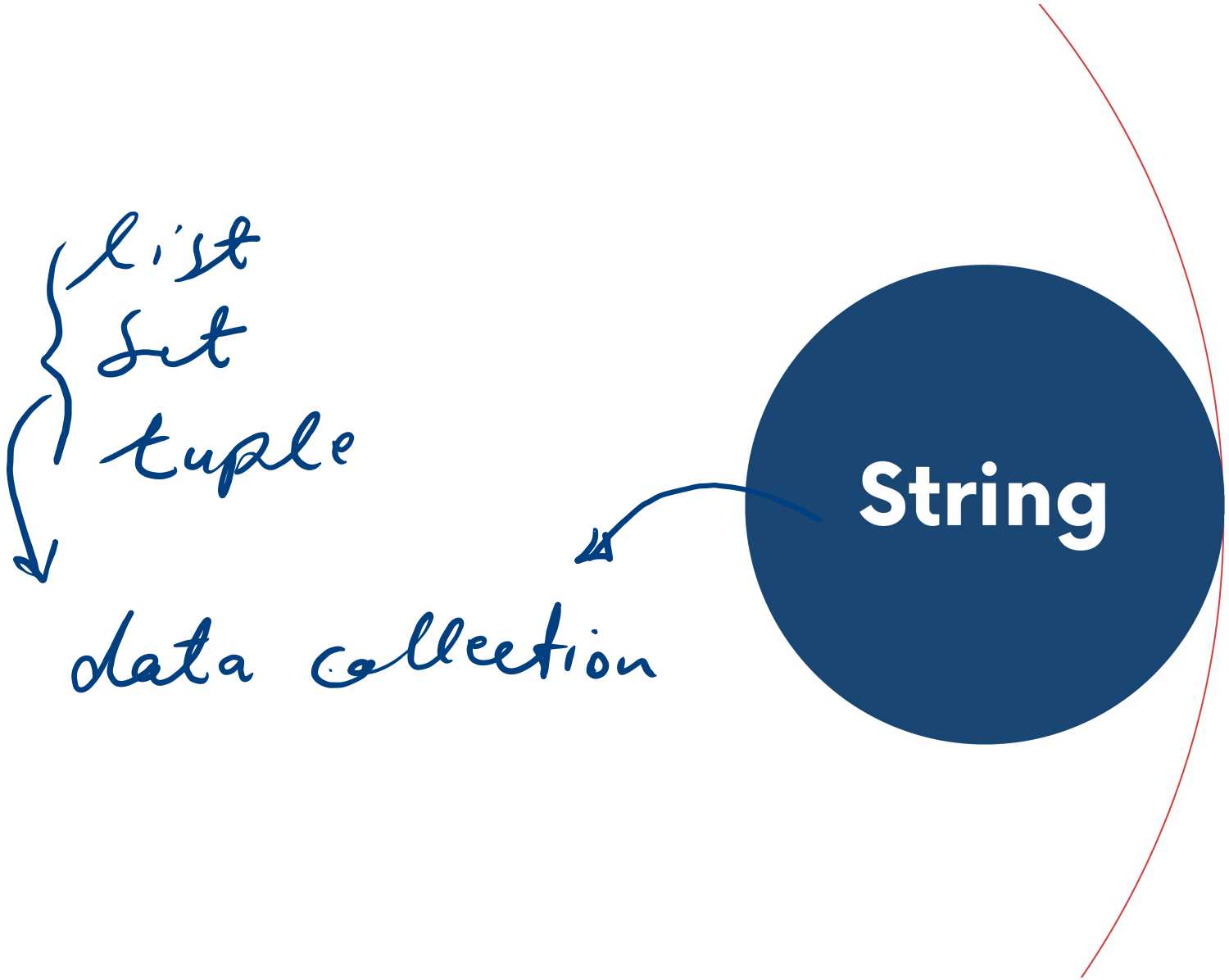

list
set
tuple
data collection
String

Strings

- think of as a sequence of case sensitive characters
- can compare strings with ==, >, < etc.

```
>>> len()  #Is a function used to retrieve the length
           #of the string in the parentheses
```

```
>>> s = "abc"
>>> len(s)  # output: 3
```

- str1 = 'Maryam'
  77
  str2 = 'Maria'
  77

---

str1 == str2

↳ False.

str1 > str2

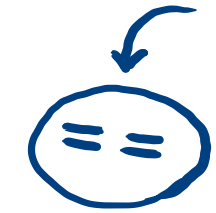
Ascii

M : 77

a : 97

Maybe y : 103

Maybe i : 99



if (str1 == str2)

⋮

Strings

square brackets used to perform indexing into a string to get the value at a certain index/position

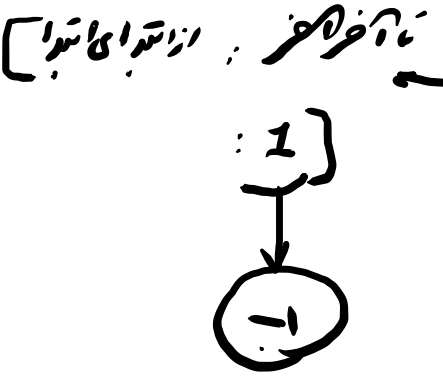
- index 0 1 2 indexing always starts at 0
- index -3 -2 -1 last element always at index -1

```
>>> s = "abc"

>>> s[0]      #out: a
>>> s[1]      #out: b
>>> s[2]      #out: c
>>> s[3]      #out: error
>>> s[-1]     #out: c
>>> s[-2]     #out: b
>>> s[-3]     #out: a
```

String Slicing

- can slice strings using [start:stop:step]
- if give two numbers, [start:stop], step=1 by default
- you can also omit numbers and leave just colons

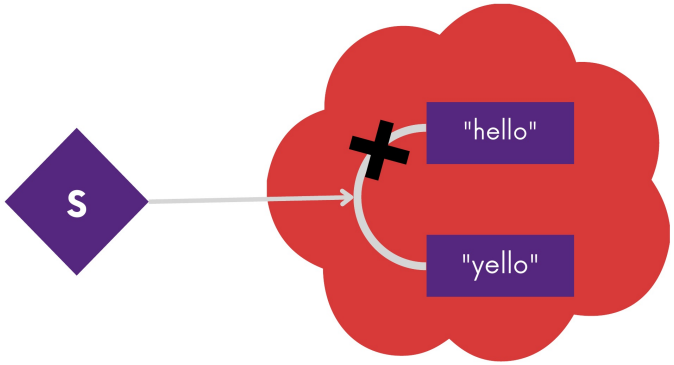
```
>>> s = "abcdefgh"

>>> s[3:6]          #out: "def", same as s[3:6:1]
>>> s[3:6:2]        #out: "df"
>>> s[::]           #out: "abcdefgh", same as s[0:len(s):1]
>>> s[::-1]         #out: "hgfedbca", same as s[-1:-(len(s)+1):-1]
>>> s[4:1:-2]       #out: "ec"
```



String Slicing

- strings are “immutable” – cannot be modified
- if give two numbers, [start:stop], step=1 by default
- you can also omit numbers and leave just colons

```
>>> s = "hello"
>>> s[0] = 'y'
#out: error
>>> s = 'y'+s[1:len(s)]
#out: "yello" , is allowed,
# s bound to new object
```



Strings & Loops

- these two code snippets do the same thing

```
>>> s = "abcdefgh"

>>> for index in range(len(s)):
>>>     if s[index] == 'i' or s[index] == 'u':
>>>         print("There is an i or u")
```

preferred

```
>>> for char in s:
>>>     if char == 'i' or char == 'u':
>>>         print("There is an i or u")
```

str. format

The f-strings provide a way to embed variables and expressions inside a string literal using a clearer syntax than the format() method.

F-strings

- F'text {first_variable} text {second_variable}' --> F or f

```
s = F'Hello, {first_name} {last_name}!'
```

- Multiline F-String

```
message = ( f'Hello {name}. '
f"You're learning Python at {website}.")

message = f'Hello {name}. ' \
f"You're learning Python at {website}."

message = f"""Hello {name}. You're
learning Python at {website}."""
```

F-strings

- Format numbers using f-strings

```
number = 200
 s = f'{number: 06}'
print(s)
#space06
# 00200, pad zeros
# at the beginning of the number
```

```
number = 9.98567
 s = f'{number: .2f}'
print(s) # 9.99
```

```
number = 400000000000
 s = f'{number: ,}'
 # also can use _
print(s) # 400,000,000,000
```

```
number = 0.1259
s = f'{number: .2%}'
print(s)
# 12.59%
s = f'{number: .1%}'
print(s) # 12.5%
```

path :

\t → tab space

\n → newline

Raw strings

In Python, when you prefix a string with the letter r or R such as r'...' and R'...', that string becomes a raw string.

```
s ='lang\tver\nPython\t3'
print(s)

lang ver
Python 3
```

```
s = r'lang\tver\nPython\t3'
print(s)

#out: lang\tver\nPython\t3
```

Backslash

- #backslash (\) escape other special characters
- \n: new line
- \t: Tab means(8 spaces)
- \\: Back slash
- \': Single quote (')
- \": Double quote (")

```
print('I suppose you are student.\nAre you ?')

#I suppose you are student.
#Are you ?
```

```
print('I suppose you are student.\\nAre you ?')

#I suppose you are student.\nAre you ?
```

```
print('Days\tTopics\tExercises')

#Days Topics Exercises
```

```
print('Days\\tTopics\\tExercises')

#Days\tTopics\tExercises
```

1 file

→ ✓

Dead line

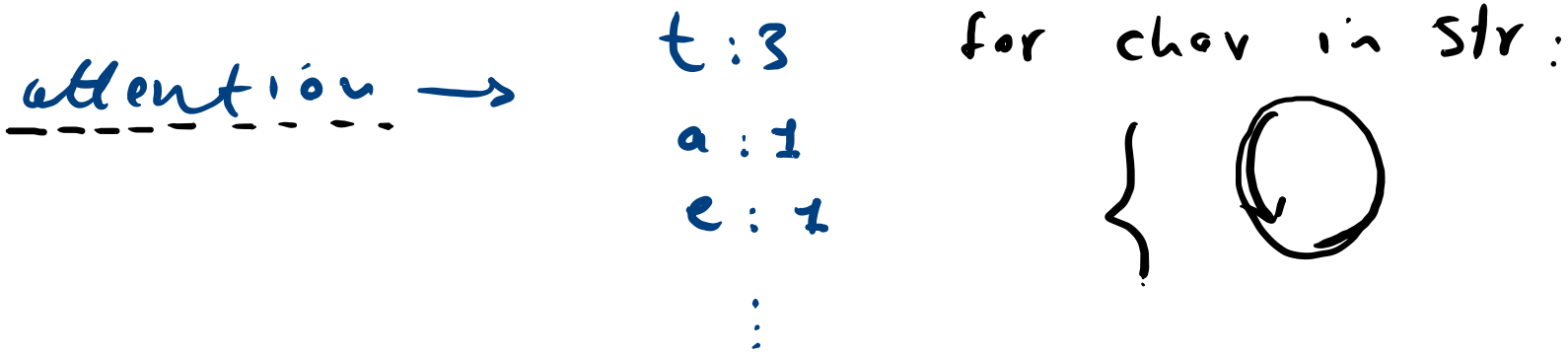
Exercises 08

String
700

String

Exercise 1

- Count the number of characters (characters frequency) in a string and print the result as a dictionary!



- Example: input: 'Filoger' , output: {'f': 1, 'i': 1, 'l': 1, 'o': 1, 'g': 1, 'e': 2, 'r': 1}

String

Exercise 2

- Replace 'Google' with the 'ChatGPT' the following text and print that again!

'Google is the border of science and innovation!'

- Example: output: 'ChatGPT is the border of science and innovation!'

String

Exercise 3

- Get the current data and time and print the output (same as example)

points:

- use **datatime** library to get the data and time
- use **f-string** to print the output!

- `output: Current date is Feb 2 2023, and the time is 14:51`

String

Exercise 4 (Search)

string_one : filoger

- How many time 'filoger' is repeated in the following text?

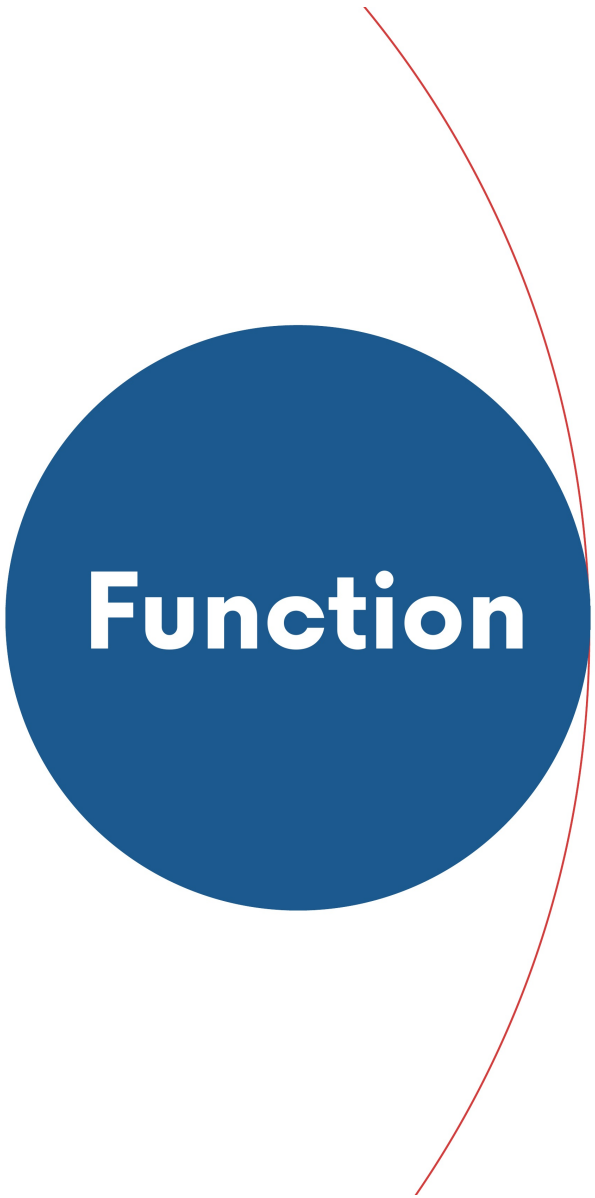
'about FILOGER: Filoger is an educational institute that focuses on AI. filoger is the biggest farsi-speakers AI community.'

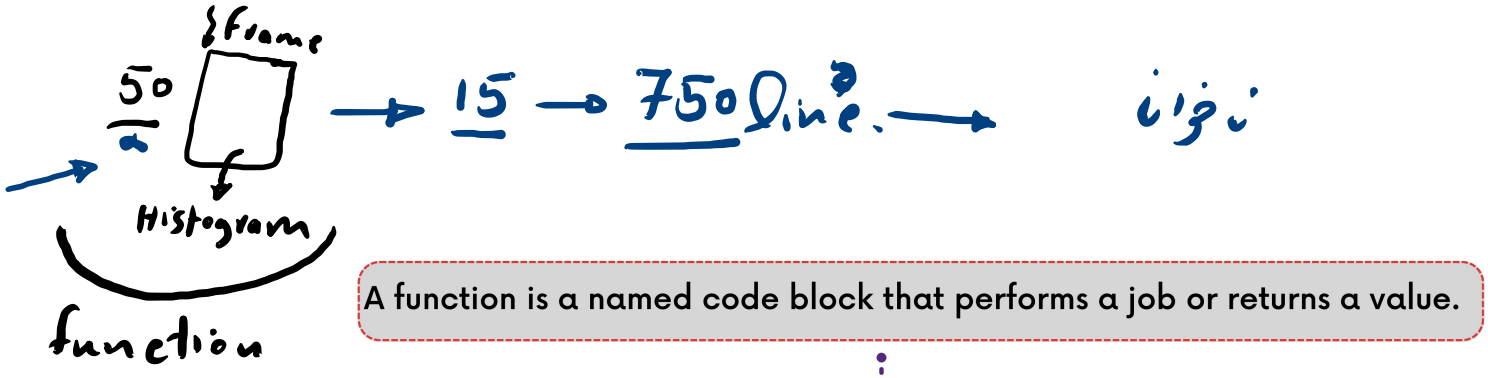
lowercase

points:

- use **f-string** to print the output!
- Uppercase and lowercase letters do not matter.

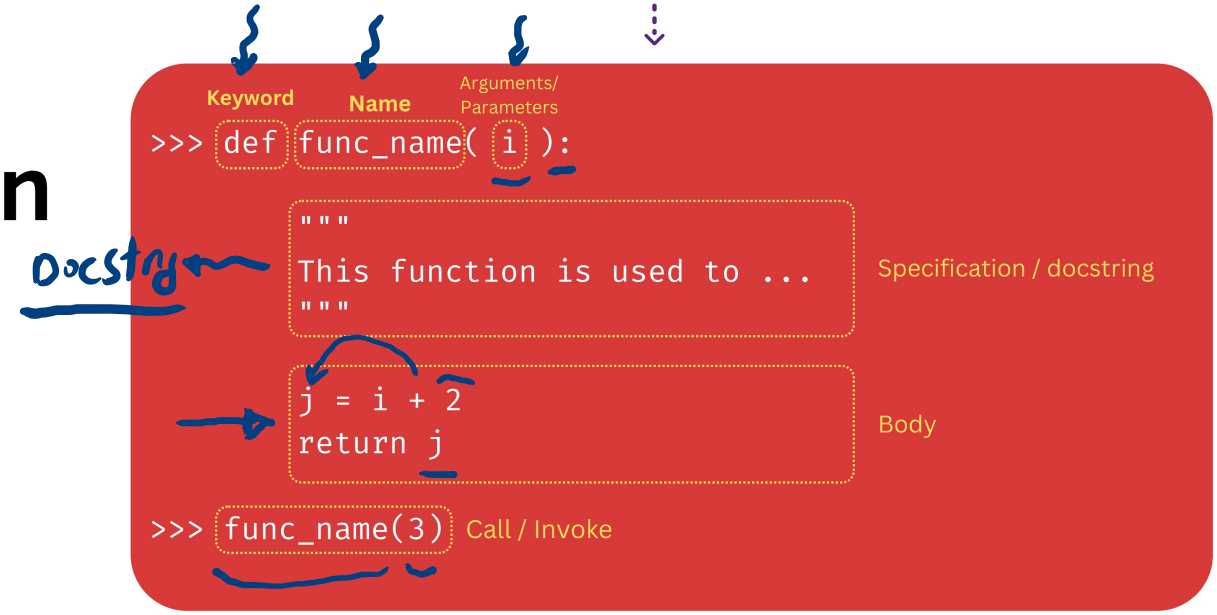
```
output: Filoger is repeated 3 times in the text!
```


Function



A function is a named code block that performs a job or returns a value.

Python Function



متغیری که داخل func تعریف شده خارج از func قابل استفاده نیست!

Variable Scope

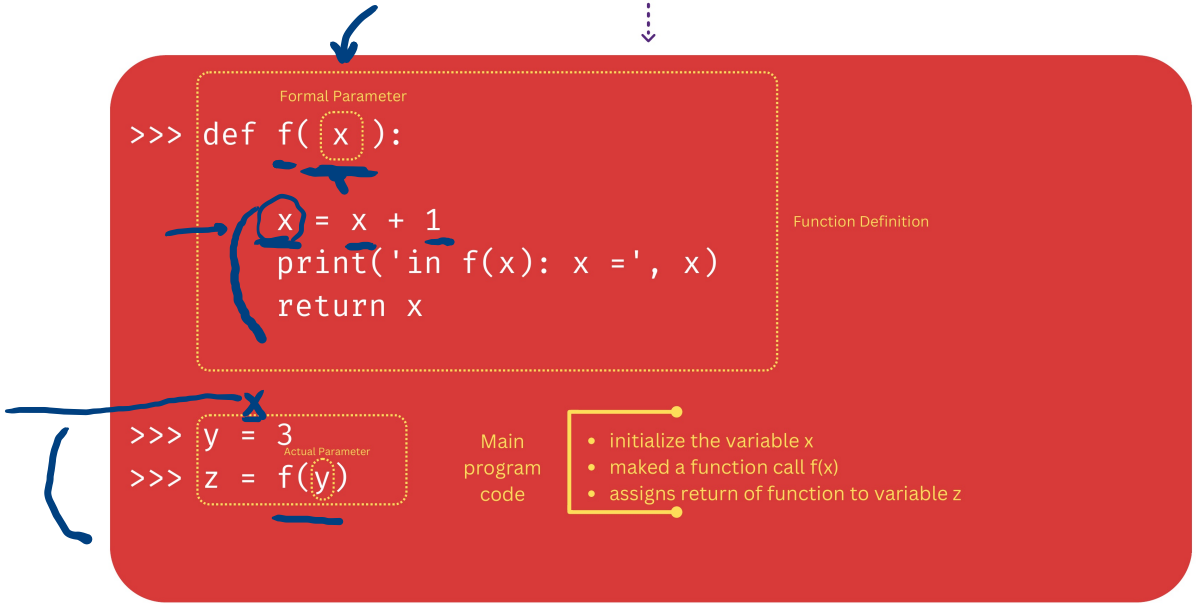
- **Scope:** A variable is only available from inside the region it is created.

src: w3schools.com

x : local var!

Local Scope

- A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.

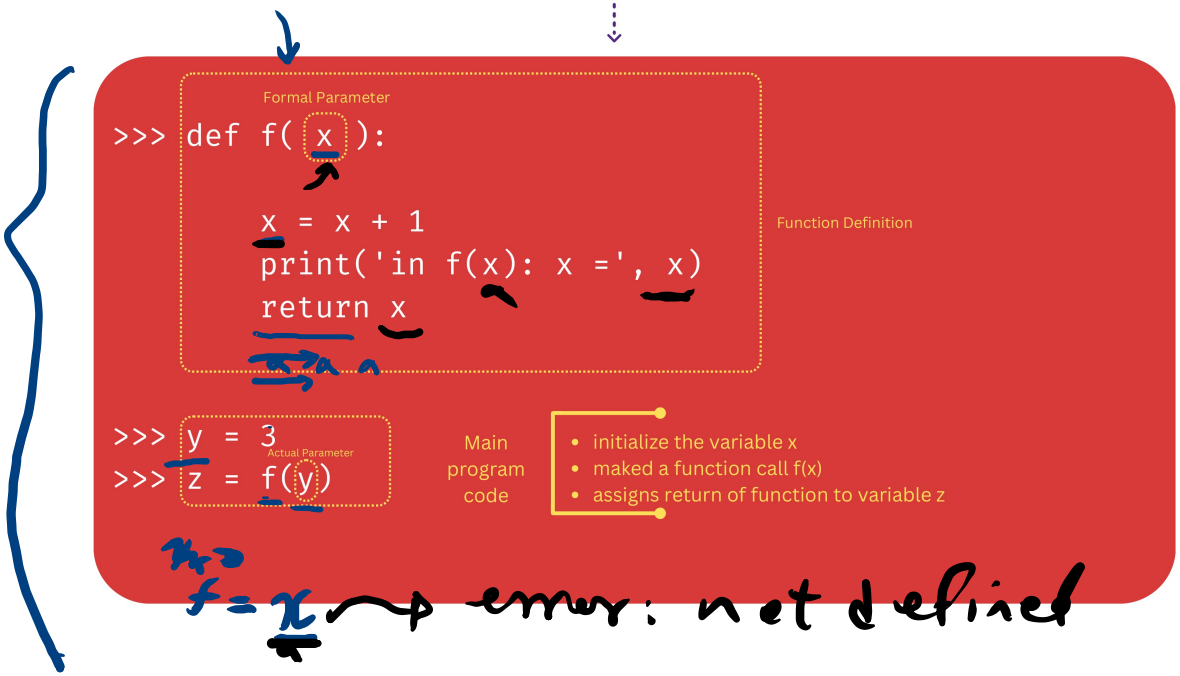


src: w3schools.com

std_size = 256 x 256.

Global Scope

- A variable created in the main body of the Python code is a global variable and belongs to the global scope.

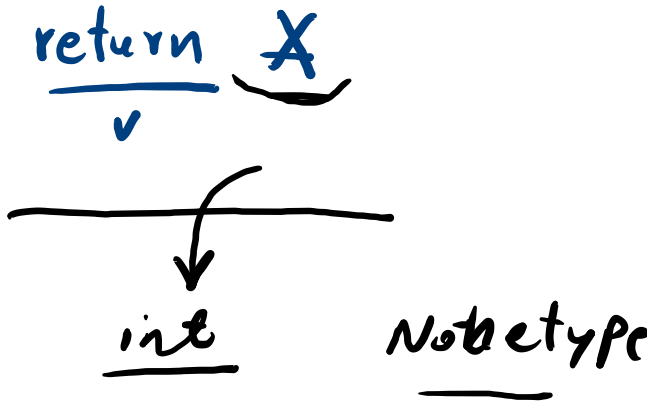


src: w3schools.com

One warning if no return statement!

```
>>> def f( x ):
        """
        Text
        """
        x = x + 1
        x%2
```

Without a return statement



- Python returns the value None, if no return given
- represents the absence of a value

Function as an Argument

fun1 (fun2)

```
#arguments can take on any type, even functions
>>> def func_a():
        print 'inside func_a'

>>> def func_b(y):
        print 'inside func_b'
        return y

>>> def func_c(z):
        print 'inside func_c'
        return z()

>>> print func_a()
>>> print 5 + func_b(2)
>>> print func_c(func_a)
```

```
def calculate_speed (frame_rate, distance):
    speed = distance x frame_rate
    return speed

cs = calculate_speed (↓ , ↓ )
```

Default Parameters

When you define a function, you can specify a default value for each parameter.

```
def function_name(param1, param2 = value, param3):
```

Default Parameter

```
def greet(name, message='Hi'):
    return f"{message} {name}"

greeting = greet('John', 'Hello')
print(greeting)
```

'Hello John'

greet('John')

'Hi John'

Default Parameters

```
def student(firstname, lastname ='Mark', standard ='Fifth'):
    print(firstname, lastname, 'studies in', standard, 'Standard')

# 1 positional argument
student('John')
#out: John Mark studies in Fifth Standard

# 3 positional arguments
student('John', 'Gates', 'Seventh')
#out: John Gates studies in Seventh Standard

# 2 positional arguments
student('John', 'Gates')
#out: John Gates studies in Fifth Standard

student('John', 'Seventh')
#out: John Seventh studies in Fifth Standard
```

default

src: geeksforgeeks.org

def func (age, name, breed):

func (81, 'Ali', 'Asian')

fun (age=81, name='Ali', breed='Asian')

```
def get_net_price(price, discount):
    return price * (1-discount)

net_price = get_net_price(0.1, 100)
print(net_price)
```

Keyword Arguments

the function call get_net_price(100, 0.1) has a readability issue. Because by looking at that function call only, you don't know which argument is price and which one is the discount.

(81, 'Ali', 'Asian')

(name = 'Ali',
age = 81, breed = 'Asian')

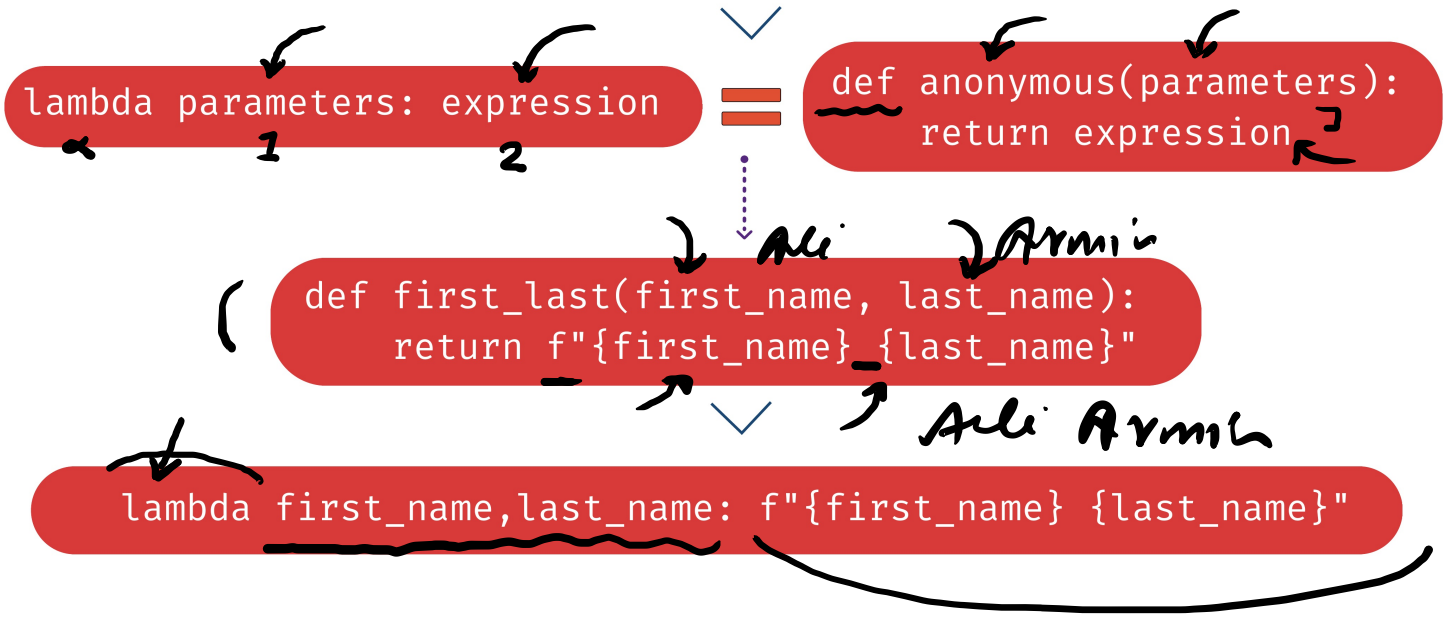
```
#fn(parameter1=value1,parameter2=value2)

net_price = get_net_price(price=100, discount=0.1)
print(net_price)
```

Lambda Expressions

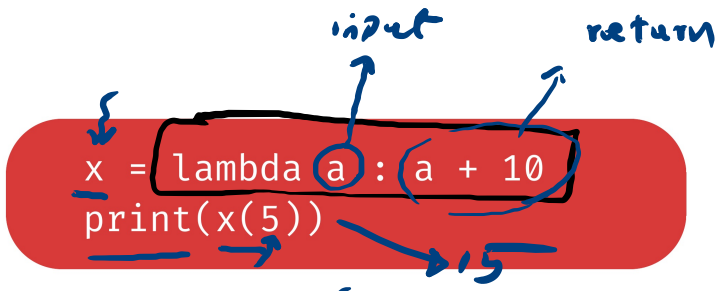
Python lambda expressions allow you to define anonymous functions.

Anonymous functions are functions without names. The anonymous functions are useful when you need to use them once.

A lambda expression typically contains one or more arguments, but it can have only one expression.



Lambda Examples




```
x = lambda a, b : a * b
print(x(5, 6))
```

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

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
print(mydoubler(11))
```

src: w3schools.com

Print vs Return

Return	Print
return only has meaning inside a function	print can be used outside functions
only one return executed inside a function	can execute many print statements inside a function
code inside function but after return statement not executed	code inside function can be executed after a print statement
has a value associated with it, given to function caller	has a value associated with it, outputted to the console

Exercises 09

Function
600

Function

Exercise 1

- Write a Python function to print the odd numbers from a given list.

- Sample List : [1, 2, 3, 4, 5, 6, 7, 8, 9], Expected Result : [1, 3, 5, 7, 9]

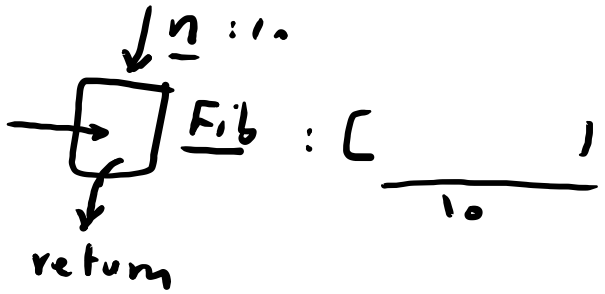
Function

Exercise 2

- Write a Python function to calculate first n numbers of Fibonacci sequence. (n is function input)

points:

- Use iteration to solve the question!



- input: 7, output: 0 1 1 2 3 5 8

Function

Exercise 3

- Write a Python function to sort a given list of dictionaries (employees) using Lambda. (Sort by age, then for same ages, sort by name)

employees = [{"name": "Sanaz", "age": 14}, {"name": "Hamed", "age": 18}, {"name": "Azam", "age": 14}, {"name": "Zahra", "age": 16}, {"name": "Shayan", "age": 18}, {"name": "Zahra", "age": 17}]

- output: [{"name": "Azam", "age": 14}, {"name": "Sanaz", "age": 14}, {"name": "Zahra", "age": 16}, {"name": "Zahra", "age": 17}, {"name": "Hamed", "age": 18}, {"name": "Shayan", "age": 18}]