Professur Technische Informatik
Prof. Dr. Wolfram Hardt

Python: Functions

Exercise 1.4

Shadi Saleh

Agenda

- Python: Functions
- Built in Functions
- User defined functions
- Maps & filters
- Reduce

Python: Functions

- A function is a block of organized, reusable code that is used to perform a single, related action
- Functions provides better modularity for your application and a high degree of code reusing.

- built-in functions:
  - print() etc.
- user-defined functions.
  - Sum()

Built in Functions

- Variety of functions for numerous purposes:
  - Our programs comprise a single function called main().
  - Functions from the standard libraries (math.sqrt)

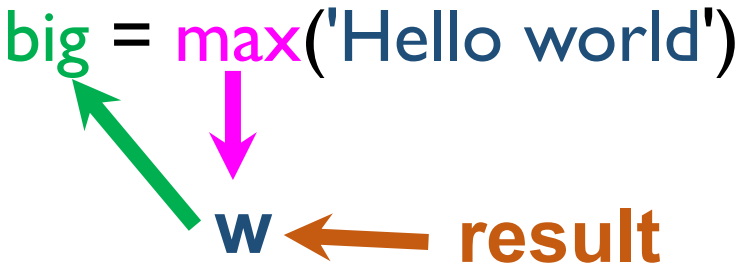
```
import math              # importing complete math package
print(math.sqrt(4))      # built in function for finding square root
s = 2*pi                 # value of pi is defined by package
print(s)
```

Output:

```
2
6.312
```

Built-in function

- Max() to find the highest element




```
big = max('Hello world')
print big

tiny = min(1, 2, 3)
print tiny
```

Type conversion

- When you put an integer and floating point in an expression the integer is implicitly converted to a float
- You can control this with the built in functions int() and float()

```
a= int (100  / 23)
print
Output:
            4
```

```
i = 42.1
 type(i)
Output:
            float
```

```
a=float (1 + 2 * 3 / 4)
print
Output:
            2.5
```

Function: rules

- Rules
  - Function blocks begin with the keyword **def** followed by the function name and parentheses ( ).
  - Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
  - The code block within every function starts with a colon (:) and is indented.
  - The statement return [expression] exits a function, optionally passing back an expression to the caller.
- Syntax:

```
def functionname( parameters ):
"function_docstring" function_suite return [expression]
```

Syntax

- Syntax:

```
def functionname( parameters ):
   "function_docstring"
    function_suite
    return [expression]
```

- By default, parameters have a positional behavior, and you need to inform them in the same order that they were defined.
- Example:

```
def printme( str ):
   print (str)          #This prints a passed string function
   return
```

Calling a function

- Following is the example to call printme() function:

```
def printme( str ):          #This is a print function
    print str;
    return;

printme("I'm first call to user defined function!");
printme("Again second call to the same function");
```

- This would produce following result:

```
I'm first call to user defined function!
Again second call to the same function
```

Example

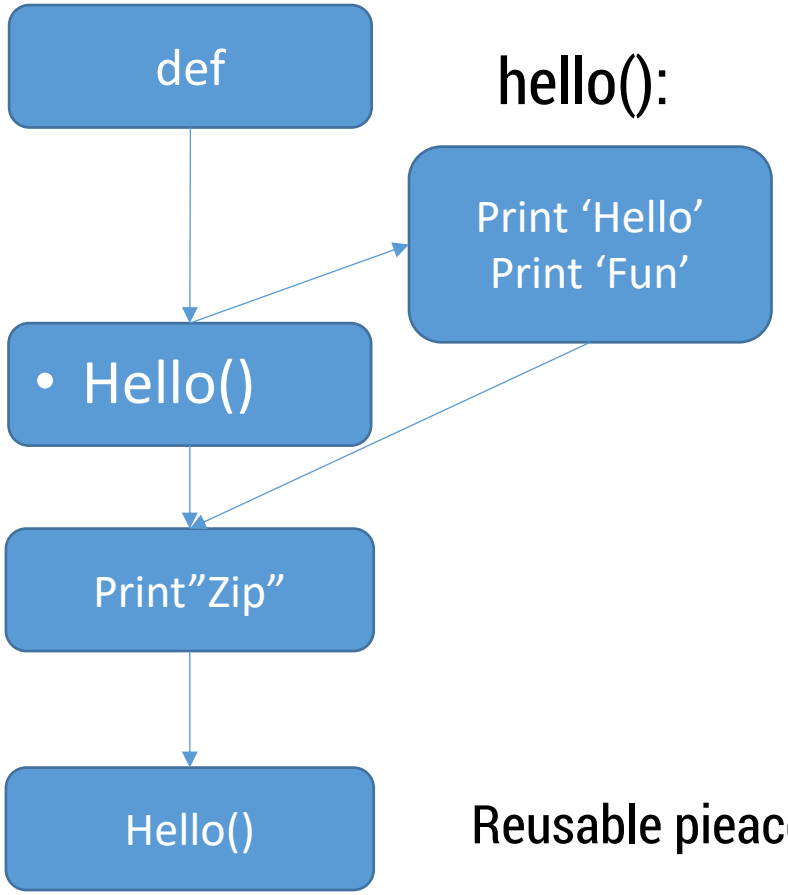
- A function is like a *subprogram*, a small program inside of a program.
- The basic idea – we write a sequence of statements and then give that sequence a name. We can then execute this sequence at any time by referring to the name.

```
def happy():  # Function defining
    print("Happy Birthday to you!")
happy() # function calling
```

Output

Happy Birthday to you!

Function: Basic structure



hello():

Print ‘Hello’
Print ‘Fun’

Program:

```
def thing():
    print 'Hello'
    print 'Fun'

thing()
print 'Zip'
thing()
```

Output:

Hello
Fun
Zip
Hello
Fun

Reusable pieaces of code “function”

Example

- Parameters:
  - A parameter is a variable which we use in the function definition that is a “handle” that allows the code in the function to access the arguments for a particular function invocation.

Function definition

```
def greet(lang):
    if lang == 'es':
        print 'Hola’
    elif lang == 'fr':
        print 'Bonjour’
    else:
        print 'Hello’
```

Function calling

```
greet('en’)
greet('es’)
greet('fr')
```

Output

```
Hello
Hola
Bonjour
```

Multiple parameters

- We can define more than one parameter in the function definition
- We simply add more arguments when we call the function
- We match the number and order of arguments and parameters

```
def addtwo(a, b):       #two parameters
    added = a + b
    # print (added)     # It will print the result but we cannot use it later in the program
    return added

x = addtwo(4, 2)
print (x)
```

User defined functions con..

- Parameters can be properly named or unnamed.
- Default values can be used

```
def say_hello(speaker, person_to_greet, greeting = "Hello"):
  print(f'{greeting}{ person_to_greet}, this is{ speaker}')

say_hello('osama'," john")
say_hello('osama',"john", "Goodbye")
say_hello(speaker = 'osama', person_to_greet = "john", greeting = "Goodbye")
```

Output:

```
Hello john, this isosama
Goodbyejohn, this isosama
Goodbyejohn, this isosama
```

User defined functions con..

- Function, can be used to trim white space from a string capitalize the first letter.

```
def process_string(str):
    t= str.strip()
    return t[0].upper()+t[1:]
str = process_string(" hi friends ")
print(f'"{str}"')
```

Output:

"Hi friends"

Python: maps

- Map function takes a list and applies a function to each member of the list and returns a second list that is the same size as the first.

```
l  = [ 'cat', 'dog', 'horse' ]
list (map(process_string, l ))
Output:
        [ 'cat', 'dog', 'horse' ]
```

- Map function is very similar to the Python comprehension

```
l  = ['cat', 'dog', 'horse']
l2 = [process_string(x) for x in l]
print (l2)
Output:
        [ 'cat', 'dog', 'horse' ]
```

Python: Filter

- Unlike map function, filter function creates a potentially smaller list.
- Filters the required elements

```
def greater_than_seven(a):
    return a>5

l= [1, 10, 20, 8, 3, -2, 0]
l2 = list(filter(greater_than_seven, l))
print(l2)
```

Output:

```
[10, 20, 8]
```

Python: Lambda

- Lambda check the values, greater or smaller
  - Separate functions can be avoided to check the values

```
l = [ 1,  10,   20,   3,   -2,   0]
l2 = list(filter(lambda x: x>5, l))
print(l2)
```

Output

```
[10, 20]
```

Scope of variables

- The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python:
  - Global variables
  - Local variables
- Global vs. Local variables:
  - Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.
  - This means that local variables can be accessed only inside the function in which they are declared whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope.

Exercise: 1

- Write a Python function to sum all the numbers in a list.

Exercise: 2

- Write a Python function to check whether a number is in a range of 3 and 9.

Thank you for your attention

Shadi Saleh

Professur Technische Informatik
Fakultät für Informatik

Technische Universität Chemnitz
Straße der Nationen 62
09111 Chemnitz
Germany

shadi.saleh@informatik.tu-chemnitz.de