UEH UNIVERSITY College of Technology and Design

Python for Data Science

By: Võ Văn Hải

Email: vovanhai@ueh.edu.vn

1

Objectives

- Python Function
- Python Modules
- Python Package
- Python Exception Handling

2

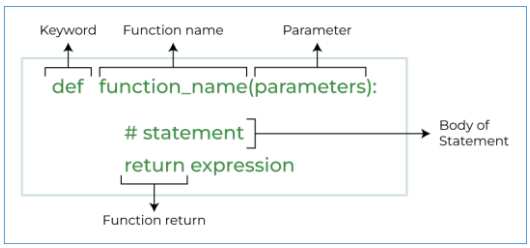
Python Function

3

Python Function

Introduction

- Python Functions is a block of statements that return the specific task.
  - The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.
- Python Function Declaration



Syntax of Python Function Declaration

- Types of Functions in Python
  - Built-in library function: These are Standard functions in Python that are available to use.
  - User-defined function: We can create our own functions based on our requirements.

4

Python Function

- Creating a Function in Python

```
# A simple Python function
def fun():
    print("Welcome to Python")
```

Calling a Function in Python

```
# Driver code to call a function
fun()
```

- Python Function Arguments
  - Arguments are the values passed inside the parenthesis of the function. A function can have any number of arguments separated by a comma.
- Example

```
# A simple Python function to check
# whether x is even or odd
def evenOdd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")
```

```
# Driver code to call the function
evenOdd(2)
evenOdd(3)
```


5

Python Function

Types of Python Function Arguments (1/2)

- Argument types in Python:
  - Default argument
  - Keyword arguments (named arguments)
  - Positional arguments
  - Arbitrary arguments (variable-length arguments *args and **kwargs)

Default Arguments

```
# Python program to demonstrate  default arguments
def myFun(x, y=50):
    print("x: ", x)
    print("y: ", y)

# Driver code (We call myFun() with only  argument)
myFun(10)
```

Keyword Arguments

```
# Python program to demonstrate Keyword Arguments
def student(firstname, lastname):
    print(firstname, lastname)

# Keyword arguments
student(firstname='Teo', lastname='Nguyen Van')
student(lastname='Nguyen Van', firstname='Teo')
```

Positional Arguments

```
def nameAge(name, age):
    print("Hi, I am", name)
    print("My age is ", age)

# Driver code
nameAge("Teo", 27)
```


6

Python Function

Types of Python Function Arguments (2/2)

- Arbitrary Keyword Arguments
- In Python Arbitrary Keyword Arguments, *args, and **kwargs can pass a variable number of arguments to a function using special symbols. There are two special symbols:
  - *args in Python (Non-Keyword Arguments)
  - **kwargs in Python (Keyword Arguments)

```
# Python program to illustrate
# *args for variable number of arguments
def myFun(*argv):
    for arg in argv:
        print(arg)

myFun('Hello', 'Welcome', 'to', 'Python world')
```

```
Hello
Welcome
to
Python world
```

output

```
# Python program to illustrate
# *kwargs for variable number of keyword arguments
def myFun(**kwargs):
    for key, value in kwargs.items():
        print("%s == %s" % (key, value))

# Driver code
myFun(first='University', mid='of', last='Ecocomics')
```

```
first == University
mid == of
last == Economics
```


Python Function

Python Function with Parameters ***for Python 3.5 and above***

- Python allows us to specify the *data type of arguments* and *return type* of the function
- Syntax

```
def function_name(parameter: data_type) -> return_type:
"""Docstring"""
# body of the function
return expression
```

```
def add(num1: int, num2: int) -> int:
    """Add two numbers"""
    num3 = num1 + num2
    return num3
```

```
# Driver code
num1, num2 = 5, 15
ans = add(num1, num2)
print(f"The addition of {num1} and {num2} results {ans}.")
```

Python Function

Docstring

- The first string after the function is called the Document string or Docstring in short.
  - This is used to describe the functionality of the function.
  - The use of docstring in functions is optional but it is considered a good practice.

```
def evenOdd(x):
    """Function to check if the number is even or odd"""
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")

# Driver code to call the function
print(evenOdd.__doc__)
```

```
# Driver code to call the function
print(evenOdd.__doc__)
```

exp.src.tam.X

```
def evenOdd(x: {__mod__}) -> None
```

Function to check if the number is even o

odd

Python Function

Python Function within Functions

- A function that is defined inside another function is known as the inner function or nested function.
- Nested functions can access variables of the enclosing scope. Inner functions are used so that they can be protected from everything happening outside the function.

```
# Python program to
# demonstrate accessing of
# variables of nested functions
def f1():
    s = 'I love UEH'
    def f2():
        print(s)
    f2()

# Driver's code
f1()
```

Python Function

Anonymous Functions in Python

- In Python, an anonymous function means that a function is without a name. As we already know the def keyword is used to define the normal functions, and the lambda keyword is used to create anonymous functions.

```
# Python code to illustrate the cube of a number
# using lambda function
def cube(x): return x * x * x

cube_v2 = lambda x: x * x * x

print(cube(7))
print(cube_v2(7))
```


Python Function

Return Statement in Python Function

- The function return statement is used to exit from a function and go back to the function caller and return the specified value or data item to the caller.
- The syntax for the return statement is:

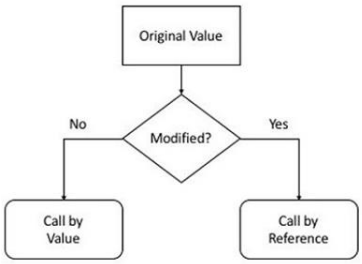
```
return [expression_list]
```

```
def find_square(num):
  # code
  return result
        2
square = find_square(3)
# code
                    1
```

Python Function

Pass by Reference and Pass by Value

- Passing arguments
  - **pass by value** – When a variable is passed to a function while calling, the value of actual arguments is copied to the variables representing the formal arguments. Thus, any changes in formal arguments does not get reflected in the actual argument. This way of passing variable is known as call by value.
  - **pass by reference** – In this way of passing variable, a reference to the object in memory is passed. Both the formal arguments and the actual arguments (variables in the calling code) refer to the same object. Hence, any changes in formal arguments does get reflected in the actual argument.




```
def swap(a, b):
    t = a
    a = b
    b = t
    return a, b

x, y = 5, 7
swap(x, y)
print(x, y)  # --> x=5,y=7
x, y = swap(x, y)
print(x, y)  # -->x=7,y=5
```

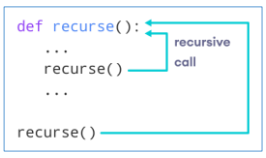
```
def inc(lst):
    for i in range(len(lst)):
        lst[i] = lst[i] + 1


arr = list(range(5))
print(arr)  # [0, 1, 2, 3, 4]
inc(arr)  # [1, 2, 3, 4, 5]
print(arr)
```


Python Function

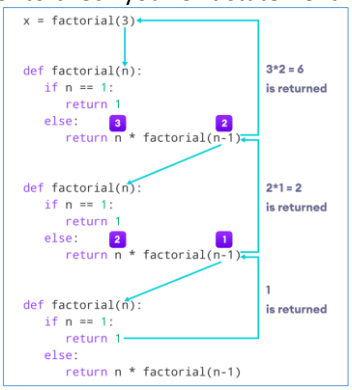
Recursive Functions in Python

- Recursion in Python refers to when a function calls itself. There are many instances when you have to build a recursive function to solve Mathematical and Recursive Problems.
- Using a recursive function should be done with caution, as a recursive function can become like a non-terminating loop. It is better to check your exit statement while creating a recursive function.




```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)


print(factorial(4))  # -->24
```



Python Function

Exercises

1. Write a Python function to find the maximum of three numbers.
2. Write a Python function to sum all the numbers in a list.
3. Write a Python program to reverse a string.
4. Write a Python function to calculate the factorial of a number (a non-negative integer). The function accepts the number as an argument.
5. Write a Python function that takes a number as a parameter and checks whether the number is prime or not.
6. Write a Python function to print
   1. all prime numbers that less than a number (enter prompt keyboard).
   2. the first N prime numbers
7. Write a Python function to check whether a number is "Perfect" or not. Then print all perfect number that less than 1000.
8. Write a Python function to check whether a string is a pangram or not.

   (Note : Pangrams are words or sentences containing every letter of the alphabet at least once. For example : "The quick brown fox jumps over the lazy dog"

15

Python Modules, Package

16

Python Modules

Introduction

- As our program grows bigger, it may contain many lines of code. Instead of putting everything in a single file, we can use modules to separate codes in separate files as per their functionality. This makes our code organized and easier to maintain.
- Module is a file that contains code to perform a specific task. A module may contain variables, functions, classes etc.
- Example

```
# Python Module addition: Save file as example.py
def add(a, b):
    result = a + b
    return result
```

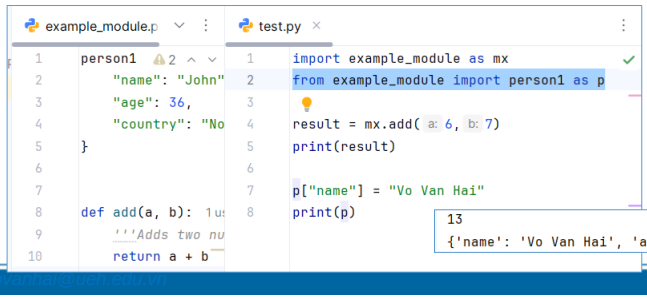
we have defined a function *add()* inside a module named *example*.

Import modules in Python

```
import example
example.add(4,5) # returns 9
```


Python Modules

Python import with Renaming



Python Modules

Import Python Standard Library Modules (built-in modules)

```
>>> help('modules')

IPython             _weakrefset         heapq               secrets
__future__          _winapi             hmac                select
_abc                abc                 html                selectors
_ast                aifc                http                setuptools
_asyncio            antigravity         idlelib             shelve
_bisect             argparse            imaplib             shlex
_blake2             array               imghdr              shutil
_bootlocale         ast                 imp                 signal
_bz2                asynchat            importlib           simplegeneric
_codecs             asyncio             ind                 site
```

Read more at https://docs.python.org/3/py-modindex.html

Python Package

Introduction

- Python Packages are a way to organize and structure your Python code into reusable components.
  - Think of it like a folder that contains related Python files (modules) that work together to provide certain functionality.
  - Packages help keep your code organized, make it easier to manage and maintain, and allow you to share your code with others.
- Creating packages in Python allows you to organize your code into reusable and manageable modules.
  - **Create a Directory**: Start by creating a directory (folder) for your package. This directory will serve as the root of your package structure.
  - **Add Modules**: Within the package directory, you can add Python files (modules) containing your code. Each module should represent a distinct functionality or component of your package.
  - **Init File**: Include an *__init__.py* file in the package directory. This file can be empty or can contain an initialization code for your package. It signals to Python that the directory should be treated as a package.
  - **Subpackages**: You can create sub-packages within your package by adding additional directories containing modules, along with their own __init__.py files.
  - **Importing**: To use modules from your package, import them into your Python scripts using dot notation. For example, if you have a module named module1.py inside a package named *mypackage*, you will import its function like this: from mypackage.module1 import greet.
  - **Distribution**: If you want to distribute your package for others to use, you can create a *setup.py* file using Python's *setuptools* library. This file defines metadata about your package and specifies how it should be installed.

Python Package

Structure

- A Python package is a way of organizing related modules into a single directory hierarchy. A package contains a special __init__.py file (which can be empty) that distinguishes it from a regular directory and allows it to be imported as a package.
- Structure

```
mypackage/
    __init__.py
    module1.py
    module2.py
    subpackage/
        __init__.py
        submodule1.py
```

- *mypackage is the main package.*
- *__init__.py indicates that mypackage is a package.*
- *module1.py and module2.py are modules within the mypackage.*
- *subpackage is a sub-package inside mypackage, with its own __init__.py.*

Example

```
mypackage/
|
├── __init__.py
├── module1.py
└── module2.py
```

```python
# module1.py
def greet(name):
    print(f"Hello, {name}!")
```

```python
# module2.py
def add(a, b):
    return a + b
```

```python
from mypackage import module1, module2

# Using functions from module1
module1.greet("Alice")

# Using functions from module2
result = module2.add(3, 5)
print("The result of addition is:", result)
```


Python Module

Well-known Packages in industry

- Python Packages for Web frameworks
  - Flask
  - Django
  - FastAPI
  - Pyramid
  - Tornado
  - Falcon
  - …
- Python Packages for AI & Machine Learning
  - NumPy
  - Pandas
  - SciPy
  - Scikit-learn
  - TensorFlow
  - Keras
  - spaCy
  - generative
  - …
- Python Packages for GUI Applications
  - Tkinter
  - PyQT5
  - Kivy
  - PySide
  - PySImpleGUI
  - NiceGUI
  - …
- Python Packages for Game Development
  - PyGame
  - Panda3D
  - PyOpenGL
  - Arcade
  - Cocos2d
  - …

Python Exception Handling

Python Exceptions Handling

Errors

- Errors are problems in a program that causes the program to stop its execution. On the other hand, exceptions are raised when some internal events change the program's normal flow.
- Different types of exceptions in python:
  - **SyntaxError**: This exception is raised when the interpreter encounters a syntax error in the code, such as a misspelled keyword, a missing colon, or an unbalanced parenthesis.
  - **TypeError**: This exception is raised when an operation or function is applied to an object of the wrong type, such as adding a string to an integer.
  - **NameError**: This exception is raised when a variable or function name is not found in the current scope.
  - **IndexError**: This exception is raised when an index is out of range for a list, tuple, or other sequence types.
  - **KeyError**: This exception is raised when a key is not found in a dictionary.
  - **ValueError**: This exception is raised when a function or method is called with an invalid argument or input, such as trying to convert a string to an integer when the string does not represent a valid integer.
  - **AttributeError**: This exception is raised when an attribute or method is not found on an object, such as trying to access a non-existent attribute of a class instance.
  - **IOError**: This exception is raised when an I/O operation, such as reading or writing a file, fails due to an input/output error.
  - **ZeroDivisionError**: This exception is raised when an attempt is made to divide a number by zero.
  - ImportError: This exception is raised when an import statement fails to find or load a module.

Python Exceptions Handling

Exceptions

- An exception is an unexpected event that occurs during program execution.
- Exceptions are raised when the program is syntactically correct, but the code results in an error. This error does not stop the execution of the program, however, it changes the normal flow of the program.
- Errors vs. Exceptions
  - Errors represent conditions such as compilation error, syntax error, error in the logical part of the code, library incompatibility, infinite recursion, etc.
  - Errors are usually beyond the control of the programmer and we should not try to handle errors.
  - Exceptions can be caught and handled by the program.

Python Exceptions Handling

Catching Specific Exceptions in Python

- For each try block, there can be zero or more except blocks. Multiple except blocks allow us to handle each exception differently.
- The argument type of each except block indicates the type of exception that can be handled by it.

```
try:
   even_numbers = [2, 4, 6, 8]
   print(even_numbers[5])

except ZeroDivisionError:
   print("Denominator cannot be 0.")

except IndexError:
   print("Index Out of Bound.")

# Output: Index Out of Bound
```

When the IndexError exception occurs in the try block,

The ZeroDivisionError exception is skipped.

The set of code inside the IndexError exception is executed.

Python Exceptions Handling

Python try with else clause

- In some situations, we might want to run a certain block of code if the code block inside try runs without any errors.
  - For these cases, you can use the optional else keyword with the try statement.

```
try:
    num = int(input("Enter a number: "))
    assert num % 2 == 0
except:
    print("Not an even number!")
else:
    reciprocal = 1/num
    print(reciprocal)
```

```
Enter a number: 5
Not an even number!
```

```
Enter a number: 4
0.25
```

```
Enter a number: 0
Traceback (most recent call last):
  File "D:\src\tam\X.py", line 9, in <module>
    reciprocal = 1 / num
                 ~~^~~~~
ZeroDivisionError: division by zero
```

the **assert** statement in the code checks that num is an even number; if num is odd, it raises an **AssertionError**, triggering the except block.

Python Exceptions Handling

Python try...except Block

- The try...except block is used to handle exceptions in Python.
- Syntax:

```
try:
    # code that may cause exception
except:
    # code to run when exception occurs
```

- Place the code that might generate an exception inside the try block.
  - Every try block is followed by an except block.
- When an exception occurs, it is caught by the except block.
  - The except block cannot be used without the try block.
- Example

```
try:
    numerator = 10
    denominator = 0
    result = numerator / denominator
    print(result)
except:
    print("Error: Denominator cannot be 0.")
```

Python Exceptions Handling

Python try...finally

- In Python, the finally block is always executed no matter whether there is an exception or not.
- The finally block is optional. And, for each try block, there can be only one finally block.

```
try:
    numerator = 10
    denominator = 0
    result = numerator / denominator
    print(result)
except:
    print("Error: Denominator cannot be 0.")
finally:
    print("This is finally block.")
```

```
Error: Denominator cannot be 0.
This is finally block.
```

The exception is caught by the except block. And, then the finally block is executed.

Python Exceptions Handling

Exercises

- Write a Python program
  1. to handle a ZeroDivisionError exception when dividing a number by zero.
  2. that prompts the user to input an integer and raises a ValueError exception if the input is not a valid integer.
  3. that prompts the user to input two numbers and raises a TypeError exception if the inputs are not numerical.
  4. that executes an operation on a list and handles an IndexError exception if the index is out of range.
  5. that prompts the user to input a number and handles a KeyboardInterrupt exception if the user cancels the input.

Summary

Python Function

Python Modules

Python Package

Python Exception Handling

32

Thanks for your listening!

33