onnet

2021
VIE

Python Programming

www.on.net.vn
www.arrowhitech.com

odoo Gold Partner

onnet
BY AHT TECH JSC

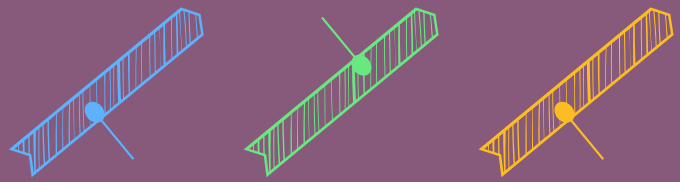


Table of Content

Before Start

- Open a Python console (online or offline) and try some codes on-the-fly
- For practical exercises, try to write it down and hand in the script after class so that I can check your syntax and give you advices
- For assignments:
  - You don’t have to do all
  - But try to do as much as u can

04

Functions

04

Functions

Function

```
# Use "def" to create new functions
def add(x, y):
    print("x is {} and y is {}".format(x, y))
    return x + y  # Return values with a return statement

# Calling functions with parameters
add(5, 6)  # => prints out "x is 5 and y is 6" and returns 11
```

04

Functions

Function

```
# Use "def" to create new functions
def add(x, y):
    print("x is {} and y is {}".format(x, y))
    return x + y  # Return values with a return statement

# Calling functions with parameters
add(5, 6)  # => prints out "x is 5 and y is 6" and returns 11

# Another way to call functions is with keyword arguments
add(y=6, x=5)  # Keyword arguments can arrive in any order.
```

04

Functions

Function

```
# You can define functions that take a variable number of
# positional arguments
def varargs(*args):
    return args

varargs(1, 2, 3)  # => (1, 2, 3)
```

04

Functions

Function

```
# You can define functions that take a variable number of
# keyword arguments, as well
def keyword_args(**kwargs):
    return kwargs

# Let's call it to see what happens
keyword_args(big="foot", loch="ness")  # => {"big": "foot", "loch": "ness"}
```

04

Functions

Function

```
# You can do both at once, if you like
def all_the_args(*args, **kwargs):
    print(args)
    print(kwargs)
"""
all_the_args(1, 2, a=3, b=4) prints:
    (1, 2)
    {"a": 3, "b": 4}
"""
```

04

Functions

Function

```
# Returning multiple values (with tuple assignments)
def swap(x, y):
    return y, x  # Return multiple values as a tuple without the parenthesis.
                 # (Note: parenthesis have been excluded but can be included)

x = 1
y = 2
x, y = swap(x, y)     # => x = 2, y = 1
# (x, y) = swap(x,y)
# Again parenthesis have been excluded but can be included.
```

04

Functions

Space Simulation - Practice 6

- Create a function launch() to check if rocket can launch successfully
- If the **chance_to_launch** > 0 return True (launched successfully)
- Else return False

$$chance\_to\_launch = 100 - 5 * \frac{cargo\_weight}{max\_weight - weight}$$

04

Functions

Function - Local Scope

```
# A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.
# The part of a program where a variable is accessible is called its scope.
def myfunc():
  x = 300
  print(x)

myfunc()
print(x)
```

04

Functions

Function - Global Scope

```
# Global variables are available from within any scope, global and local.
x = 300
def myfunc():
  print(x)

myfunc()
print(x)
```

04

Functions

Function - Global Scope

If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function):

```
x = 300
def myfunc():
  x = 200
  print(x)

myfunc()
print(x)
```

04

Functions

Function - Global Scope

```
# The global keyword makes the variable global.

def myfunc():
  global x
  x = 300

myfunc()
print(x)
```

04

Functions

Function - Global Scope

use the global keyword if you want to make a change to a global variable inside a function.

```
x = 300
def myfunc():
  global x
  x = 200

myfunc()
print(x)
```

04

Functions

First Class Function

```
# Python supports first class functions
# functions can be treat as other variables
def create_adder(x):
    def adder(y):
        return x + y
    return adder

add_10 = create_adder(10)
add_10(3)   # => 13
```

04

Functions

Higher Order Function

```
# A higher order function is a function that takes a function as an argument, or returns a function
list(map(add_10, [1, 2, 3]))         # => [11, 12, 13]
list(map(max, [1, 2, 3], [4, 2, 1]))  # => [4, 2, 3]

list(filter(lambda x: x > 5, [3, 4, 5, 6, 7]))  # => [6, 7]
```

04

Functions

Anonymous Function

```
# syntactic sugar for a normal function definition
# syntactically restricted to a single expression
(lambda x: x > 2)(3)              # => True
(lambda x, y: x ** 2 + y ** 2)(2, 1)  # => 5
```

04

Functions

List Comprehension

```
# We can use list comprehensions for nice maps and filters
# List comprehension stores the output as a list which can itself be a nested list
[add_10(i) for i in [1, 2, 3]]         # => [11, 12, 13]
[x for x in [3, 4, 5, 6, 7] if x > 5]  # => [6, 7]

# You can construct set and dict comprehensions as well.
{x for x in 'abcddeef' if x not in 'abc'}  # => {'d', 'e', 'f'}
{x: x**2 for x in range(5)}  # => {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

05
Modules

05

Modules

Import

```
# You can import modules
import math
print(math.sqrt(16))  # => 4.0

# You can get specific functions from a module
from math import ceil, floor
print(ceil(3.7))   # => 4.0
print(floor(3.7))  # => 3.0
```

05

Modules

Import

```
# You can import all functions from a module.
# Warning: this is not recommended
from math import *

# You can shorten module names
import math as m
math.sqrt(16) == m.sqrt(16)  # => True
```

05

Modules

Modules

```
# Python modules are just ordinary Python files. You
# can write your own, and import them. The name of the
# module is the same as the name of the file.

# You can find out which functions and attributes
# are defined in a module.

import math
dir(math)

# The local folder has priority over Python's built-in libraries.
```

05

Modules

Space Simulation - Practice 7

- Actually **chance_to_launch** doesn't depend on weights only
- More accurately, we should consider some randomness factor
- Re-implement launch() function (hint: use random module)

$$chance\_to\_launch = (random\ float\ from\ 0\ to\ 100) - 5 * \frac{cargo\_weight}{max\_weight - weight}$$

06
Classes

06

Classes

Class Definition

```
class Human:
    # A class attribute. It is shared by all instances of this class
    species = "H. sapiens"

    # Basic initializer, this is called when this class is instantiated.
    # Methods(or objects or attributes) like: __init__, __str__, etc.
    # are called special methods (or sometimes called dunder methods)
    # You should not invent such names on your own.
    def __init__(self, name):
        # Assign the argument to the instance's name attribute
        self.name = name
        # Initialize property
        self._age = 0
```

06

Classes

Class Definition

```
class Human:

    # An instance method. All methods take "self" as the first argument
    def say(self, msg):
        print("{name}: {message}".format(name=self.name,
                  message=msg))

    # Another instance method
    def sing(self):
        return 'yo... yo... microphone check... one two... one two...'
```

06

Classes

Class Definition

```
class Human:

    # An instance method. All methods take "self" as the first argument
    def say(self, msg):
        print("{name}: {message}".format(name=self.name,
                  message=msg))

    # Another instance method
    def sing(self):
        return 'yo... yo... microphone check... one two... one two...'
```

06

Classes

Space Simulation - Practice 8

- Create a class **Item** to represent a cargo item
- A cargo item includes **name** and its **cargo_weight**
- Store the data loaded from text file to a list of **Item** object

07

Inheritance

07

Inheritance

Inheritance

```
# Inheritance allows new child classes to be defined that inherit
methods and variables from their parent class.

# To import functions from other files use the following format
# from "filename-without-extension" import "function-or-class"

from human import Human

# Specify the parent class(es) as parameters to the class definition
class Superhero(Human):

    # Child classes can override their parents' attributes
    species = 'Superhuman'
```

07

Inheritance

Inheritance

```
class Superhero(Human):
    species = 'Superhuman'
    def __init__(self, name, movie=False,
                 superpowers=["super strength", "bulletproofing"]):

        # add additional class attributes:
        self.fictional = True
        self.movie = movie
        # be aware of mutable default values, since defaults are shared
        self.superpowers = superpowers
        # This calls the parent class constructor:
        super().__init__(name)
```

07

Inheritance

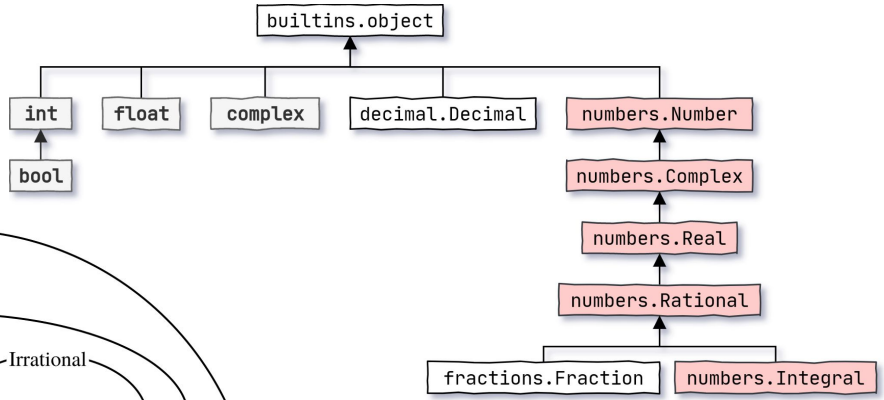
Inheritance

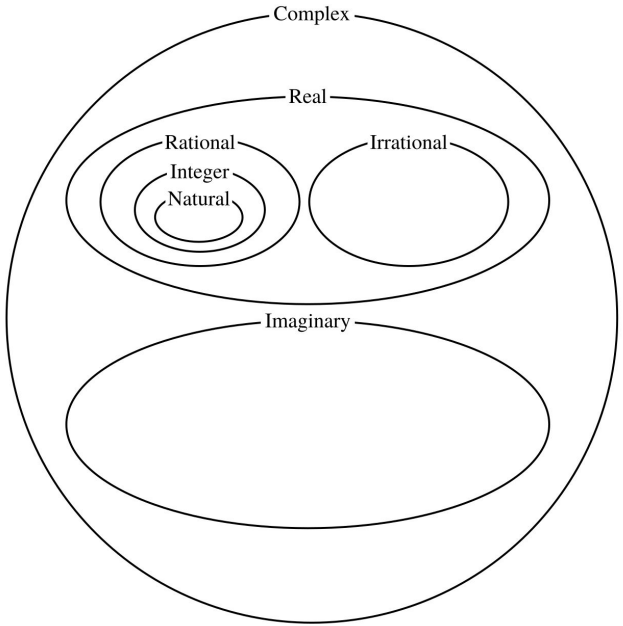
```
class Superhero(Human):
    species = 'Superhuman'
    def __init__(self, name, movie=False,
                 superpowers=["super strength", "bulletproofing"]):

        # add additional class attributes:
        self.fictional = True
        self.movie = movie
        # be aware of mutable default values, since defaults are shared
        self.superpowers = superpowers
        # This calls the parent class constructor:
        super().__init__(name)
```

07

Inheritance

Inheritance





02

Variables and Collections

Space Simulation - Practice 9

- Create a class **SpaceShip**. A spaceship can always "launch" successfully (launch() always returns True :))))
- **Rocket** is a child class of **SpaceShip** with its own **weight**, **max_weight** and **cargo_items**
- Implement function **load_item(text_file)** with input is a string of text file name
- Overwrite **launch()** function (use formula above)

07

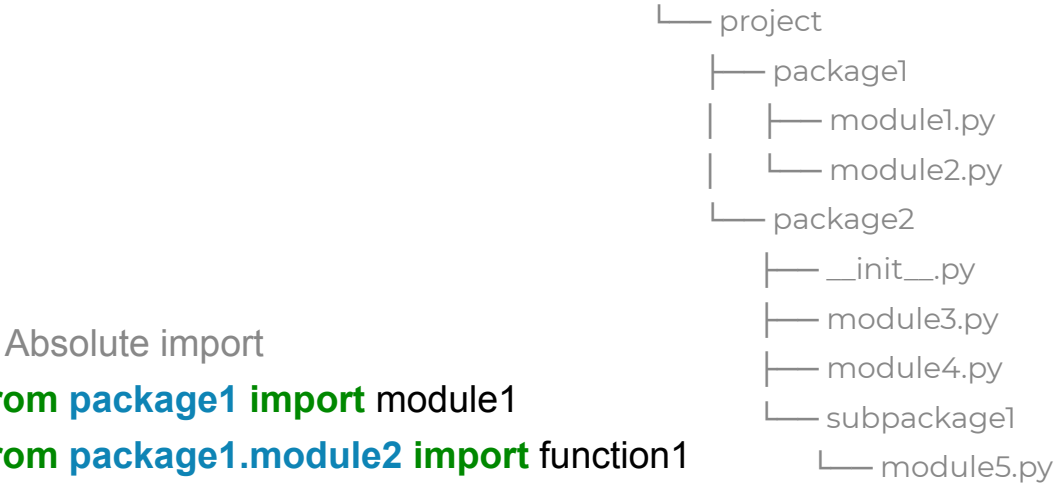
Inheritance

Multiple Inheritance

```
class Superhero(Human):
    species = 'Superhuman'
    def __init__(self, name, movie=False,
                 superpowers=["super strength", "bulletproofing"]):
class Bat:
    species = 'Baty'
    def __init__(self, can_fly=True):
                 self.fly = can_fly

class Batman(Superhero, Bat):
    def __init__(self, *args, **kwargs):
        Superhero.__init__(self, 'anonymous', movie=True,
                           superpowers=['Wealthy'], *args, **kwargs)
        Bat.__init__(self, *args, can_fly=False, **kwargs)
        self.name = 'Sad Affleck'
```

08
More

Packages

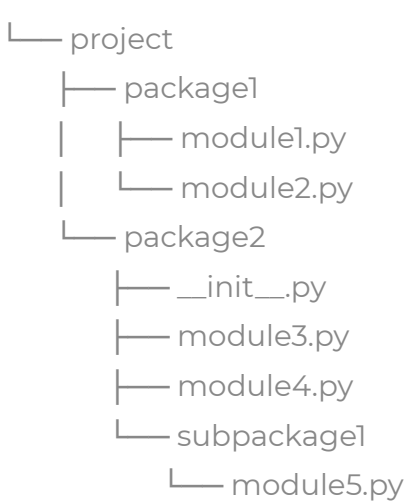



```
# Absolute import
from package1 import module1
from package1.module2 import function1
from package2 import class1
from package2.subpackage1.module5 import function2
```

08

More

Packages




```
# Relative import

# package1/module1.py
from .module2 import function1

# package2/module3.py
from .subpackage1.module5 import function2
from ..package1.module1 import function1
```

08

More

Virtual Environment

- By default, every project on your system will use these same directories to store and retrieve site packages (third party libraries)

- Python can’t differentiate between versions in the site-packages directory.

- So both v1.0.0 and v2.0.0 would reside in the same directory with the same name

08

More

Virtual Environment

- The main purpose of Python virtual environments is to create an isolated environment for Python projects.

```
$ python3 -m venv env          # create new virtualenv
$ source env/bin/activate      # activate virtualenv
(env) $
(env) $ deactivate             # deactivate virtualenv
$
```

Homework

- Day 1
- Day 2

Q&A


Thank you!