CS100 Python
Introduction to Programming

Lecture 24. OOP in Python

Fu Song

School of Information Science and Technology

ShanghaiTech University

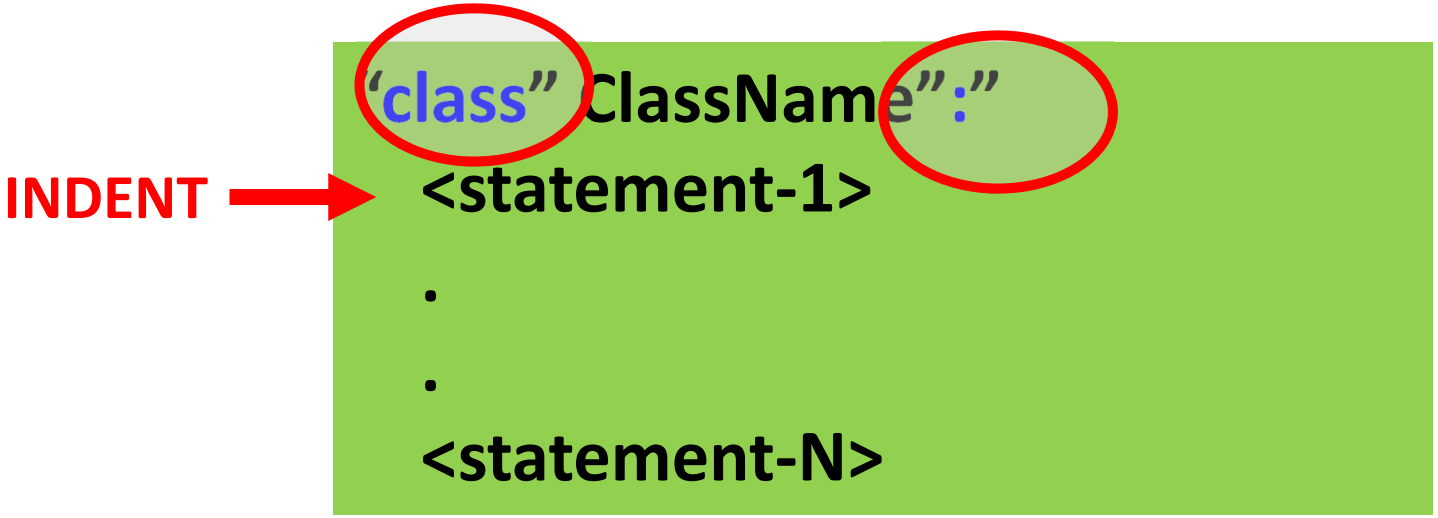
Learning Objectives

- Class
  - The smallest class
  - Constructor __init__
  - Instance attributes and Class attributes
  - Access
  - Private and Public Attributes
  - Special method names
  - Modules

Object-Oriented Programming

- In OOP, code and data are combined into a single entity called a class
  - each instance of a given class is an object of that class type
- Principles of Object-Oriented Programming
  - encapsulation
  - inheritance
  - polymorphism
- Python is Pure OO
  - Everything in Python is an object (excluding keywords)

Class Definition



- A class definition starts with the keyword class
- Following a classname, the first character of the name is usually UPPERCASE
- Then, the colon :
- The class body consists of a sequence of statements and/or function definitions, organized via INDENT

The Smallest Class

INDENT 

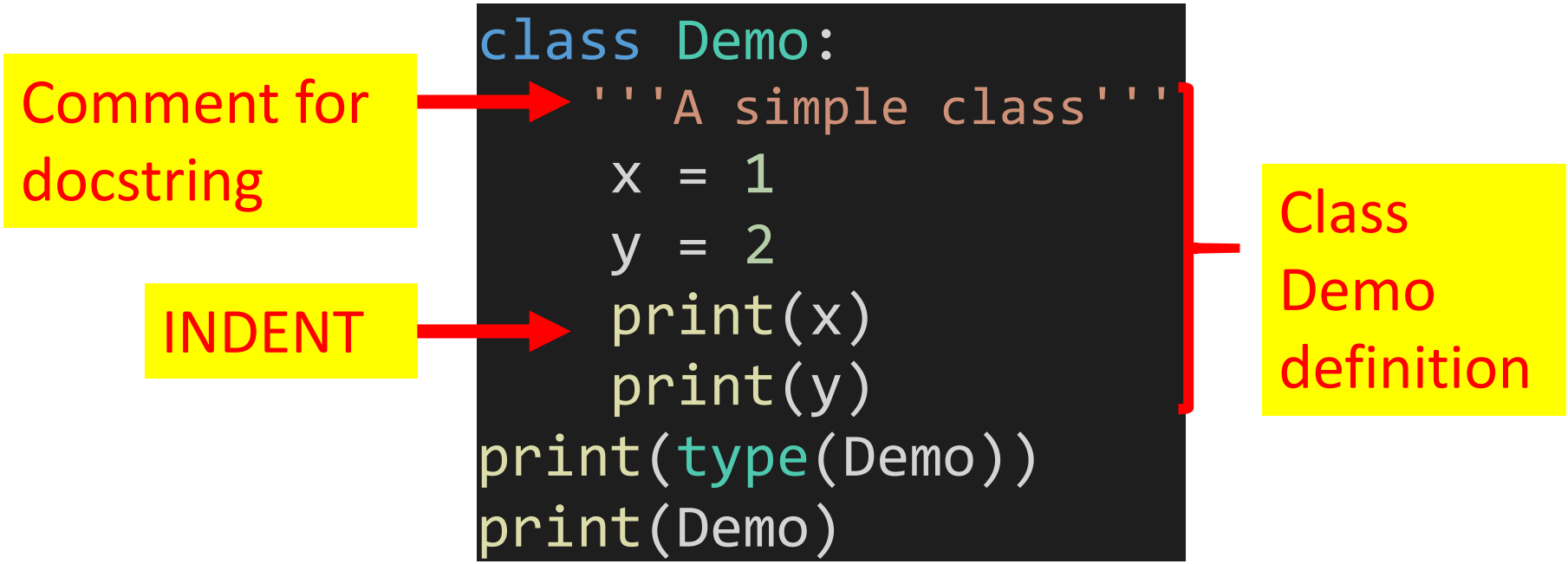
```
class Demo:
    pass
```

- Define a class, called Demo
- Only has one statement pass
- pass is a null operation when it is executed, nothing happens
- It is useful as a placeholder when a statement is required syntactically, but no code needs to be executed
- It is recommended to use 4 spaces for INDENT

Class Definition

- When a class definition is entered, a new namespace is created, and used as the local scope
- thus, all assignments to local variables go into this new namespace
- In particular, function definitions bind the name of the new function objects here.
- When a class definition is left normally (via the end), a class instance is created. This is basically a wrapper around the contents of the namespace created by the class definition
- The original scope (the one in effect just before the class definition was entered) is reinstated, and the class object is bound here to the class name given in the class definition header (ClassName in the example)
- All classes implicitly inherited the most base class object

The Smallest Class



- **Four statements are executed when enters class Demo**
- **Demo is an object/instance of the class type, called class instance**
- **The object Demo is in the global scope called __main__**

The Smallest Class

```
class Demo:
    '''A simple class'''
    x = 1
    y = 2
    print(x)
    print(y)
print(type(Demo))
print(Demo)
```

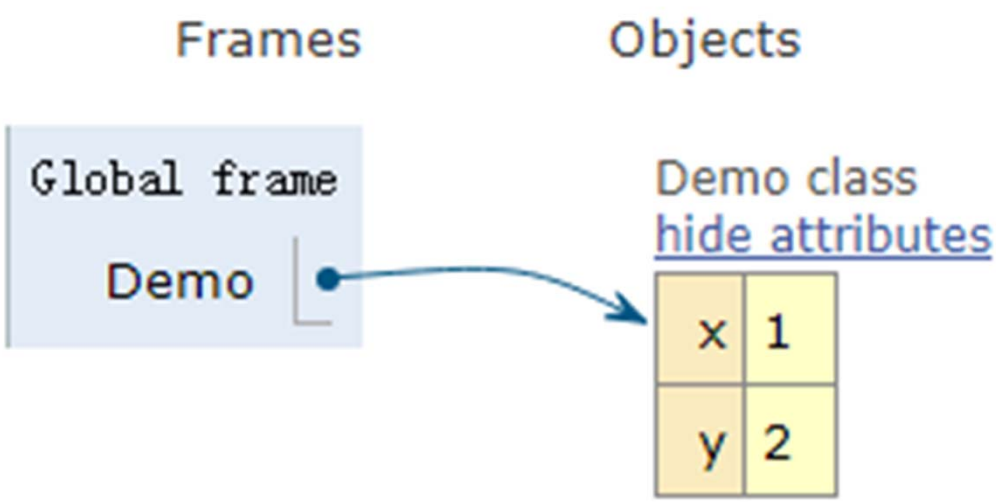
Output

```
1
2
<class 'type'>
<class '__main__.Demo'>
```

- **Four statements are executed when enters class Demo**
- **Demo is an object/instance of the class type**
- **The object Demo is in the global scope called __main__**

The Smallest Class

```
class Demo:
    x = 1
    y = 2
    print(x)
    print(y)
print(type(Demo))
print(Demo)
```



Name Demo in global namespaces is bounded to the Demo class (which is an object)

Instance objects

Class *instantiation* uses function notation:

Obj = ClassName(parameters)

```
class Demo:
    pass
print(type(Demo))
print(Demo)
d = Demo()
print(d)
print(type(d))
```

Output

```
<class 'type'>
<class '__main__.Demo'>
<__main__.Demo object at 0x02CA8490>
<class '__main__.Demo'>
```

Create an instance object of Demo

Instance objects

We can check whether an object is in instance of a class

isinstance(object, class)

```
class Demo:
    pass
d = Demo()
print(isinstance(d,Demo))
print(isinstance(Demo,type))
```

Output

```
Yes
Yes
```

Demo is a class object vs d is an instance object

Learning Objectives

- Class
  - The smallest class
  - Constructor __init__
  - Instance attributes and Class attributes
  - Access
  - Private and Public Attributes
  - Special method names
  - Modules

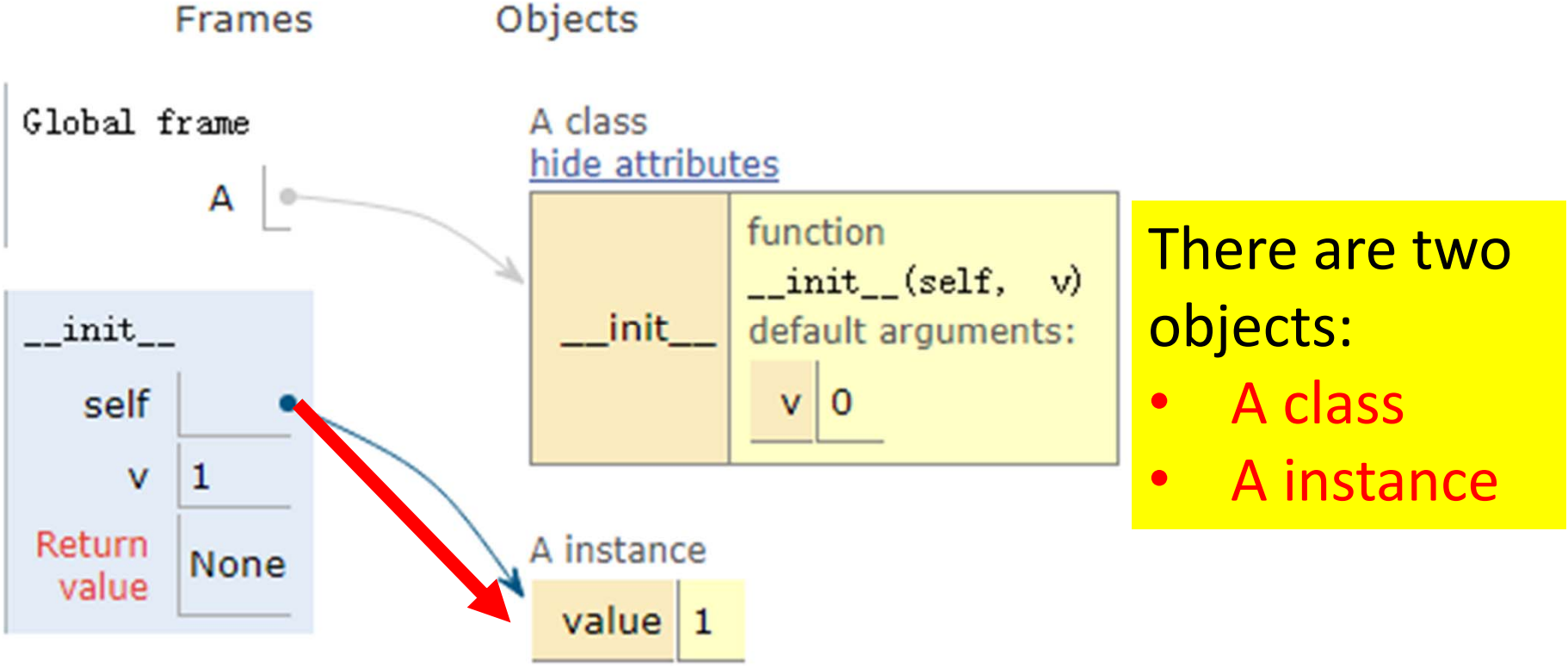
Constructor __init__

- All the classes have an implicit instance method __init__ as constructor (inherited from the class object)
- It is called after the instance has been created, but before it is returned to the caller
- The arguments are those passed to the class constructor expression
- The first parameter of __init__ is the instance object
- One can override __init__ in user-defined classes for initialization

The Smallest Class

```
class A:
    def __init__(self,v=0):
        self.value = v
a = A(1)
```

We can specify default value



There are two objects:

- A class
- A instance

self Parameter

- **In python, the first parameter of all instance methods bind to the instance object**
- **The name of the first parameter can be any identifier**
- **But, we usually use self**

```
class A:
    def __init__(self,v=0):
        self.value = v
a = A(1)
```

Self Parameter

```
class A:
    def __init__(self,v=0):
        self.value = v
a = A(1)
```

They are same

```
class A:
    def __init__(x,v=0):
        x.value = v
a = A(1)
```

Learning Objectives

- Class
  - The smallest class
  - Constructor __init__
  - Instance attributes and Class attributes
  - Access
  - Private and Public Attributes
  - Special method names
  - Modules

Attributes

- Attributes of an instance
  - instance variables: are for data unique to each instance
  - instance methods: are for manipulation of instance data
- Attributes of a class
  - class variables: are for data shared by all instances of the class
  - class methods: are for manipulation of class data
- All can be dynamically added/removed in Python
- It is better to use difference names for class attributes and instance attributes should

Instance variables

- All variables defined via

obj.var = expr

are instance variables var of the object obj

```
class A:
    def __init__(self,v=0):
        self.value = v
a = A(1)
```

value is an instance variable of the object bound to the name self

Instance variables

```
class Car:
    def __init__(self, c):
        self.color = c

car1 = Car("Red")
car2 = Car("Blue")
car1.name = "QQ"
car2.name = "BYD"
print(car1.color,car1.name)
print(car2.color,car2.name)
```

Instance variables are dynamically added into objects

Output

```
Red QQ
Blue BYD
>>>
```

Instance variables

```
class Car:
    def __init__(self, c):
        self.color = c

car1 = Car("Red")
car2 = Car("Blue")
car1.name = "QQ"
print(car1.color,car1.name)
print(car2.color,car2.name)
```

Added instance variable is specific to the object

Output

```
Red QQ
Traceback (most recent call last):
  File "D:\Test\fib.py", line 9, in <module>
    print(car2.color,car2.name)
AttributeError: 'Car' object has no attribute 'name'
```

Instance variables

```
class Car:
    def __init__(self, c):
        self.color = c

car1 = Car("Red")
car2 = Car("Blue")
car1.name = "QQ"
car2.name = "BYD"
del car2.color
print(car1.color,car1.name)
print(car2.color,car2.name)
```

An instance variable is deleted from the object

Output

```
Red QQ
Traceback (most recent call last):
  .....
AttributeError: 'Car' object has no attribute 'color'
```

Instance variables

It is recommended to initialize all the instance variables in the constructor __init__

Class variables

- All variables defined via

var = expr

in the class definition are instance variables var of the class object

```
class A:
    value = "classvariable"
    def __init__(self):
        self.value = "instancevariable"
a = A()
print(a.value)
print(A.value)
```

Class variables

```
class A:
    value = "classvariable"
    def __init__(self):
        self.value = "instancevariable"
    value2 = "classvariable2"

A.value3 = "AddValue"
print(A.value)
print(A.value2)
print(A.value3)
```

Defined in different places

A class variable is dynamically added

Output

```
classvariable
classvariable2
AddValue
>>>
```

Class variables

```
class A:
    value = "classvariable"
    def __init__(self):
        self.value = "instancevaria-ble"

print(A.value)
del A.value
print(A.value)
```

A class variable is dynamically deleted

Output

```
classvariable
Traceback (most recent call last):
  ....
AttributeError: type object 'A' has no attribute 'value'
```

Class variables

- It is recommended to initialize all the class variables at the beginning of the class definition

```
class A:
    value = "Good"
    def __init__(self):
        self.value = "instancevariable"
    value2 = "Bad"

A.value3 = "Worse"
```

Instance methods

- The first parameter of instance methods are the instance object, i.e., self

```
class A:
    def __init__(self,v=0):
        self.value = v
    def GetValue(self):
        return self.value
```

__init__ and GetValue are instance methods

Instance methods

SetValue is a normal function, not an instance method

```
class A:
    def __init__(self,v=0):
        self.value = v
    def GetValue(self):
        return self.value
def SetValue(self,v):
    self.value = v
a = A(1)
a.SetValue = SetValue
print(a.SetValue)
a.SetValue(a,2)
```

a.SetValue is an instance variable, not an instance method

Called with the object as first argument

Output

```
<function SetValue at 0x03032150>
>>>
```

Instance methods

```
import types
class A:
    def __init__(self,v=0):
        self.value = v
    def GetValue(self):
        return self.value
def SetValue(self,v):
    self.value = v
a = A(1)
a.SetValue = types.MethodType(SetValue, a)
print(a.SetValue)
a.SetValue(2)
```

import module types

Dynamically add **SetValue** as an instance method of the object **a**

Direct call SetValue via the object a

Output

```
<bound method SetValue of
<__main__.A object at 0x02FE8470>>
```

Instance methods

```
class A:
    def __init__(self,v=0):
        self.value = v
    def GetValue(self):
        return self.value
a = A(1)
print(a.GetValue)
del a.GetValue
print(a.GetValue)
```

A instance method is dynamically deleted

Output

```
<bound method GetValue of <__main__.A object at
0x035984F0>>
Traceback (most recent call last):
.....
AttributeError: 'A' object has no attribute ‘GetValue'
```

Instance Methods

- It is recommended to define all the instance methods in class definition

Class methods

- The first parameter of class methods are the class object, i.e., cls

```
class A:
    ClassValue = 1
    def __init__(self,v=0):
        self.value = v
    @classmethod
    def GetClassValue(cls):
        return cls.ClassValue

print(A.GetClassValue)
```

@classmethod is a Decorator claiming that the function defined following this is a class method

Output

```
<bound method A.GetClassValue of
<class '__main__.A'>>
```

cls Parameter

- In python, the first parameter of all class methods bind to the class object
- The name of the first parameter can be any identifier
- But, we usually use ~~clf~~ `cls`

Class methods

```
import types
class A:
    ClassValue = 1
    def __init__(self,v=0):
        self.value = v
    @classmethod
    def GetClassValue(cls):
        return cls.ClassValue
def SetClassValue(cls,v):
        cls.ClassValue = v
A.SetClassValue = types.MethodType(SetClassValue,A)
print(A.SetClassValue)
```

Dynamically add **SetClassValue** as a class method of the class **A**

Output

```
<bound method SetClassValue of
<class '__main__.A'>>
```

Class methods

```
class A:
    ClassValue = 1
    def __init__(self,v=0):
        self.value = v
    @classmethod
    def GetClassValue(cls):
        return cls.ClassValue
print(A.GetClassValue)
del A.GetClassValue
print(A.GetClassValue)
```

GetClassValue of the class **A** is deleted

Output

```
<bound method A.GetClassValue of <class
'__main__.A'>>
Traceback (most recent call last):
  ...
AttributeError: type object 'A' has no attribute 'GetClassValue'
```

Learning Objectives

- Class
  - The smallest class
  - Constructor __init__
  - Instance attributes and Class attributes
  - Access
  - Private and Public Attributes
  - Special method names
  - Modules

Access

- **Instance variables: are accessed via**

  object.var

- **Instance methods: are accessed via**

  object.f(p_1,…,p_n) or class.f(object,p_1,…,p_n)

- **Class variables: are accessed via**

  class.var or object.var

- **Class methods: are accessed via**

  class.f(p_1,…,p_n) or object.f(p_1,…,p_n)

Assuming all attributes are distinct

Access

- **Instance variables: are accessed via**

  object.var

- **Instance methods: are accessed via**

  object.f(p_1,…,p_n)

- **Class variables: are accessed via**

  class.var

- **Class methods: are accessed via**

  class.f(p_1,…,p_n)

It is better to use these forms

Access Instance attributes

```
class Car:
    def __init__(self, c):
        self.color = c
    def GetColor(self):
        return self.color
car = Car("Red")
print(car.color)
print(car.GetColor())
print(Car.GetColor(car))
print(Car.color)
```

Output

```
Red
Red
Red
Traceback (most recent call last):
....
    print(Car.color)
AttributeError: type object 'Car' has no attribute 'color'
```

self ?

Access Class attributes

```
class Car:
    color = "Blue"
    @classmethod
    def GetColor(cls):
        return cls.color
car = Car()
print(car.color)
print(Car.color)
print(car.GetColor())
print(Car.GetColor())
print(GetColor(Car))
```

Output

```
Blue
Blue
Blue
Blue
Traceback (most recent call last):
...
   print(GetColor(Car))
NameError: name 'GetColor' is not defined
```

Learning Objectives

- Class
  - The smallest class
  - Constructor __init__
  - Instance attributes and Class attributes
  - Access
  - Private and Public Attributes
  - Special method names
  - Modules

Private and Public Attributes

- In Python, there is no keywords

  public, private, friend, protected

- Python uses **underscore** to define special attributes
  - ✓ _xxx: denotes protected attribute xxx which cannot be imported using 'from module import *'
  - ✓ __xxx__: system defined attribute xxx, e.g., __init__
  - ✓ __xxx: private attribute xxx, which should be accessed via instance methods, cannot be accessed via object.__xxx outside of the class, or instance methods of its subclasses (we can still access via "object._class__xxx")
- Note: Python does not have strict private attribute

Private and Public Attributes

```
class Car:
    def __init__(self, c):
        self.__color = c
    def GetColor(self):
        return self.__color

car = Car("Red")
print(car.GetColor())
print(Car.GetColor(car))
print(car.__color)
```

__init__: special name method

__color: intended to be private instance attribute

__color: can be accessed in instance methods

AttributeError

Output

```
Red
Red
Traceback (most recent call last):
AttributeError: 'Car' object has no attribute '__color'
```

Private and Public Attributes

```
class Car:
    def __init__(self, c):
        self.__color = c
    def __GetColor(self):
        return self.__color

car = Car("Red")
print(car.__GetColor())
```

__color and __GetColor: intended to be private attribute

AttributeError

Output

```
Traceback (most recent call last):
  File "C:\Users\Fu Song\Desktop\hello.py", line 8, in
<module>
    print(car.__GetColor())
AttributeError: 'Car' object has no attribute '__GetColor'
```

Private and Public Attributes

```
class Car:
    def __init__(self, c):
        self.__color = c
    def __GetColor(self):
        return self.__color

car = Car("Red")
print(car._Car__color)
print(car._Car__GetColor())
```

__color and __GetColor: intended to be private attribute

But, they can be accessed via special way

Output

```
Red
Red
>>>
```

Private and Public Attributes

```
class Car:
    __color ="Blue"
    @classmethod
    def GetColor(cls):
        return cls.__color

print(Car.GetColor())
print(Car.__color)
```

__color: intended to be private class attribute

__color: can be accessed in this class methods

AttributeError

Output

```
Blue
Traceback (most recent call last):
  ...
AttributeError: type object 'Car' has no attribute '__color'
```

Private and Public Attributes

```
class Car:
    __color ="Blue"
    @classmethod
    def __GetColor(cls):
        return cls.__color

print(Car.__GetColor())
```

__GetColor: intended to be private class attribute

AttributeError

Output

```
Traceback (most recent call last):
  File "C:\Users\Fu Song\Desktop\hello.py", line 7, in
<module>
    print(Car.__GetColor())
AttributeError: type object 'Car' has no attribute '__GetColor'
```

Private and Public Attributes

```
class Car:
    __color ="Blue"
    @classmethod
    def __GetColor(cls):
        return cls.__color
print(Car._Car__color)
print(Car._Car__GetColor())
```

But, they can be accessed via special way

Output

```
Blue
Blue
<<<
```

Learning Objectives

- Class
  - The smallest class
  - Constructor __init__
  - Instance attributes and Class attributes
  - Access
  - Private and Public Attributes
  - Special method names
  - Modules

Special method names

- A class can implement certain operations that are invoked by special syntax (such as constructor, destructor) by defining methods with special names
- This is Python's approach to operator overloading, allowing classes to define their own behavior with respect to language operators
- They methods can be overridden if needed
- However, they methods should not be called explicitly in program
- They are called by the interpreter

Constructor and Destructor

- **Creator __new__(cls,…):** is called to create a new instance of class cls, e.g.,

  $$\mathbf{obj=cls(p_1,\ldots,p_n)}$$

  - ✓ __new__ is a static method that takes the class of which an instance was requested as its first argument. The other arguments are those passed to __init__(self,…)
  - ✓ __new__ returns the new object instance after __init__ returns

- **Constructor __init__(self,…):** is called after the instance has been created to initialize instance variables

- **Destructor __del__(self):** is called when the instance is about to be destroyed, the object is destroyed when the reference count of the object reaches zero

Example

```
class Account:
    NumofAccounts = 0

    def __init__(self,idNum,v = 0):
        assert v>=0
        self.idNum = idNum
        self.balance = v
        Account.NumofAccounts +=1

    def Deposit(self,v):
        assert v>0
        self.balance += v

    def Withdraw(self,v):
        assert 0<v<=self.balance
        self.balance -= v
```

可以直接 Num of Accounts 记为

Example

```
    def __del__(self):
        assert Account.NumofAccounts>=1
        Account.NumofAccounts -=1

    def GetBalance(self):
        print("Balance of ",
              self.idNum, "is:",self.balance)

    @classmethod
    def GetNumofAccounts(cls):
        print("Number of accounts is:",
              cls.NumofAccounts)
```

Example

```
a = Account(1,10)
b = Account(2,20)
c = Account(3,30)
a.GetBalance()
b.GetBalance()
c.GetBalance()
Account.GetNumofAccounts()
a = None
Account.GetNumofAccounts()
b = None
Account.GetNumofAccounts()
```

Output

```
Balance of  1 is: 10
Balance of  2 is: 20
Balance of  3 is: 30
Number of accounts is: 3
Number of accounts is: 2
Number of accounts is: 1
Number of accounts is: 0
```

Common special method names

Method	Description
__new__()	Create a new object instance
__init__()	Constructor
__del__()	Destructor
__add__()	+
__sub__()	-
__mul__()	*
__truediv__()	/
__floordiv__()	//
__mod__()	%
__pow__()	**
__eq__()、 __ne__()、 __lt__()、 __le__()、 __gt__()、 __ge__()	==、 !=、<、 <=、>、 >=
__lshift__()、__rshift__()	<<、>>
__and__()、__or__()、 __invert__()、__xor__()	&、|、~、^
__str__()	string representation of an object

Example

Implement a class for rational number

- Support

  ‘+’, ‘-’ ,’*’, ‘/’ ,’pow’,….,

- Does not support

  ‘and’, ‘or’,….

Example

```
def gcd ( a, b ):
    '''Return the greatest common divisor
       of a and b
    '''
    if b == 0:
        return a
    else:
        return gcd(b, a%b)
```

Euclidean Algorithm: gcd(a,b) = gcd(b,a%b)

Example

```
class Rational:
    """An instance represents a
       rational number."""

    def __init__(self,n=0, d=1):
        """Constructor for Rational."""
        assert d!=0, "d cannot be zero."
        g = gcd (n, d)
        self.n = int(n/g)
        self.d = int(d/g)
        __and__ = None
        __or__ = None
        # list non-supported methods
```

求最大公约数

It is better to assign non-supported methods by None

Example

```
    def __add__(self, other):
        """Add two rational numbers."""
        return Rational(self.n * other.d +
                        other.n * self.d,
                        self.d * other.d )

    def __sub__(self,other):
        """Return self minus other."""
        return Rational (self.n * other.d -
                         other.n * self.d,
                         self.d * other.d )
    def __str__ (self):
        """Display self as a string."""
        return str(self.n)+"/"+str(self.d)
```

Example

```
r1 = Rational(2,4)
r2 = Rational(1,4)
print(r1)
print(r1-r2)
print(r1+r2)
print(r1&r2)
```

Output

1/2
1/4
3/4
Traceback (most recent call last):
…
TypeError: unsupported operand type(s) for &:
'Rational' and 'Rational'

Learning Objectives

- Class
  - The smallest class
  - Constructor __init__
  - Instance attributes and Class attributes
  - Access
  - Private and Public Attributes
  - Special method names
  - Modules

Modules

- As your program gets longer,
  - You may want to split it into several files for easier maintenance.
  - You may also want to use a handy function that you've written in several programs without copying its definition into each program
- Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a **module**
- Definitions from a module can be **imported** into other modules or into the main module
- The concept of modules is added into C++20

Modules

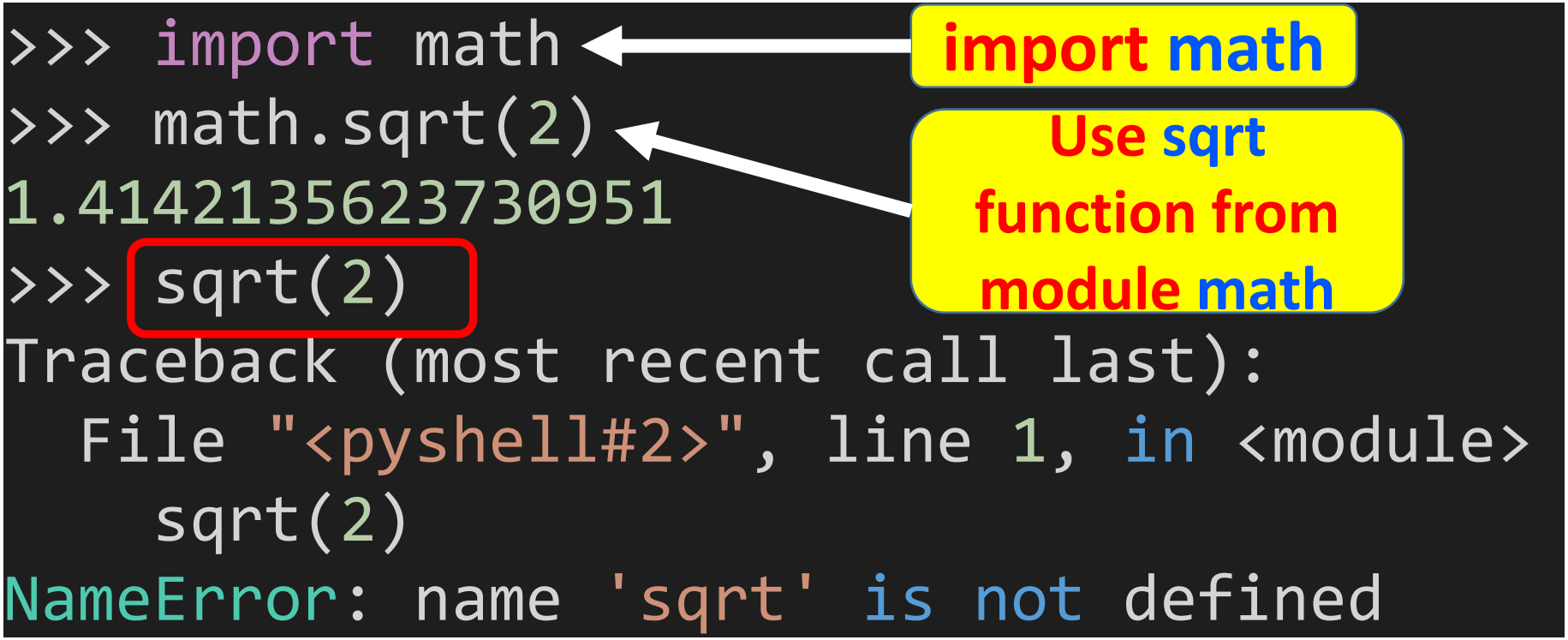
- A module is a file containing Python definitions and statements
- The file name is the module name

  ModuleName.py
- Within a module, the module's name (as a string) is available as the value of the global variable __name__
- To use a module,

  import ModuleName
- To access names in the module,

  ModuleName.Name

Modules


>>> import math
>>> math.sqrt(2)
1.4142135623730951
>>> sqrt(2)
Traceback (most recent call last):
File "<pyshell#2>", line 1, in <module>
sqrt(2)
NameError: name 'sqrt' is not defined
import math
Use sqrt function from module math

Modules

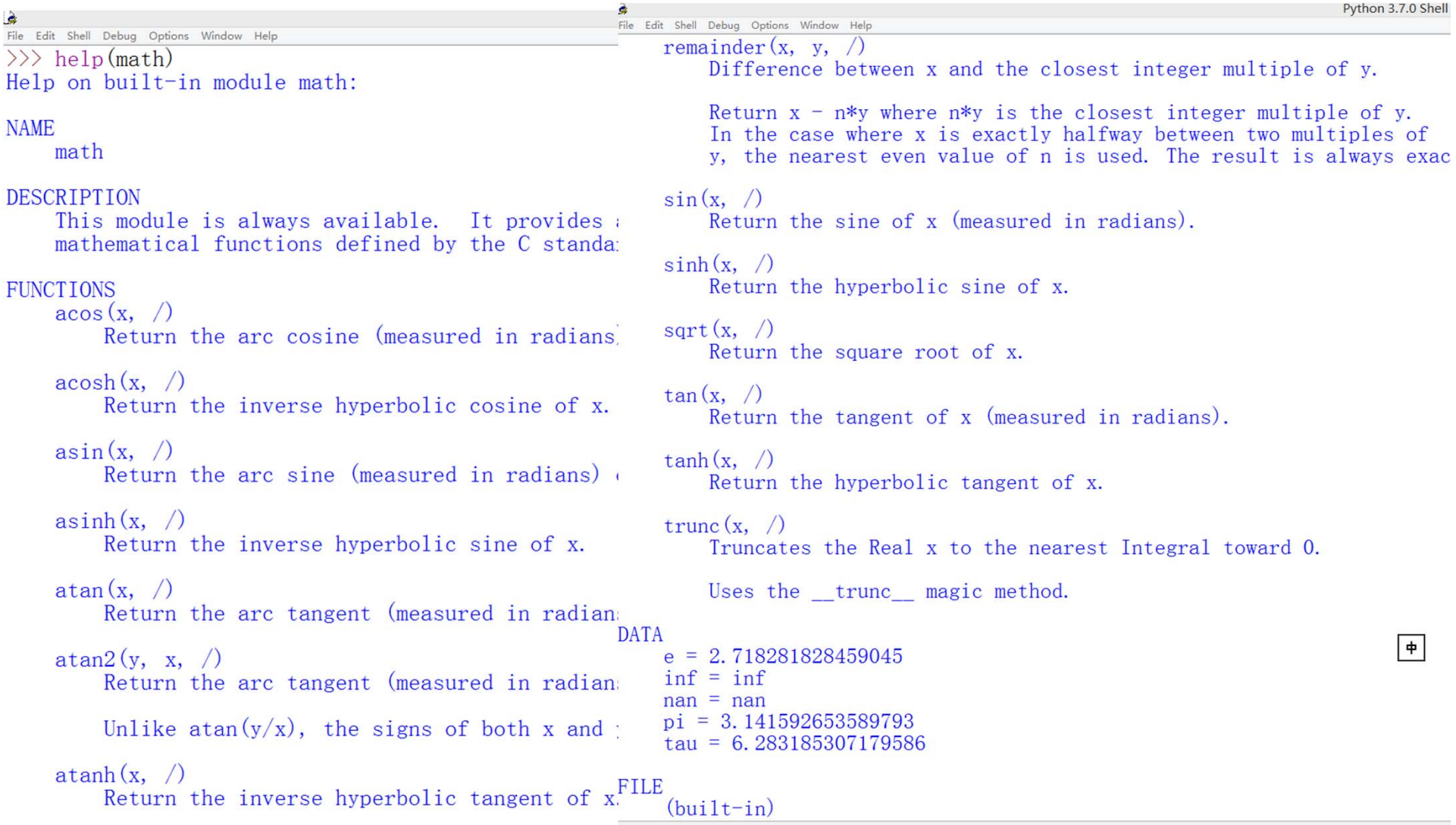
- To list all the names in an module and how to use these names

```
import moduleName
dir(moduleName)
help(moduleName)
```

- Import only needed names via

```
from ModuleName import n1,n2,...,nk
```

Help(math)



These are “’comments”’ in the math.py produced by Docstring

Modules

There are lots of modules in Python

1. Compiled-in modules: list all compiled-in module names via the sys module

```
import sys
sys.builtin_module_names
```

2. All built-in modules:

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

Compiled-in Modules

```
>>> import sys
>>> sys.builtin_module_names
('_abc', '_ast', '_bisect', '_blake2', '_codecs', '_codecs_cn', '_codecs_hk', '_codecs_iso2022', '_codecs_jp', '_codecs_kr', '_codecs_tw', '_collections', '_csv', '_datetime', '_functools', '_heapq', '_imp', '_io', '_json', '_locale', '_lsprof', '_md5', '_multibytecodec', '_opcode', '_operator', '_pickle', '_random', '_sha1', '_sha256', '_sha3', '_sha512', '_signal', '_sre', '_stat', '_string', '_struct', '_symtable', '_thread', '_tracemalloc', '_warnings', '_weakref', '_winapi', 'array', 'atexit', 'audioop', 'binascii', 'builtins', 'cmath', 'errno', 'faulthandler', 'gc', 'itertools', 'marshal', 'math', 'mmap', 'msvcrt', 'nt', 'parser', 'sys', 'time', 'winreg', 'xxsubtype', 'zipimport', 'zlib')
>>>
```

Modules

There are lots of modules in Python

1. Compiled-in modules: list all compiled-in module names via the sys module

```
import sys
sys.builtin_module_names
```

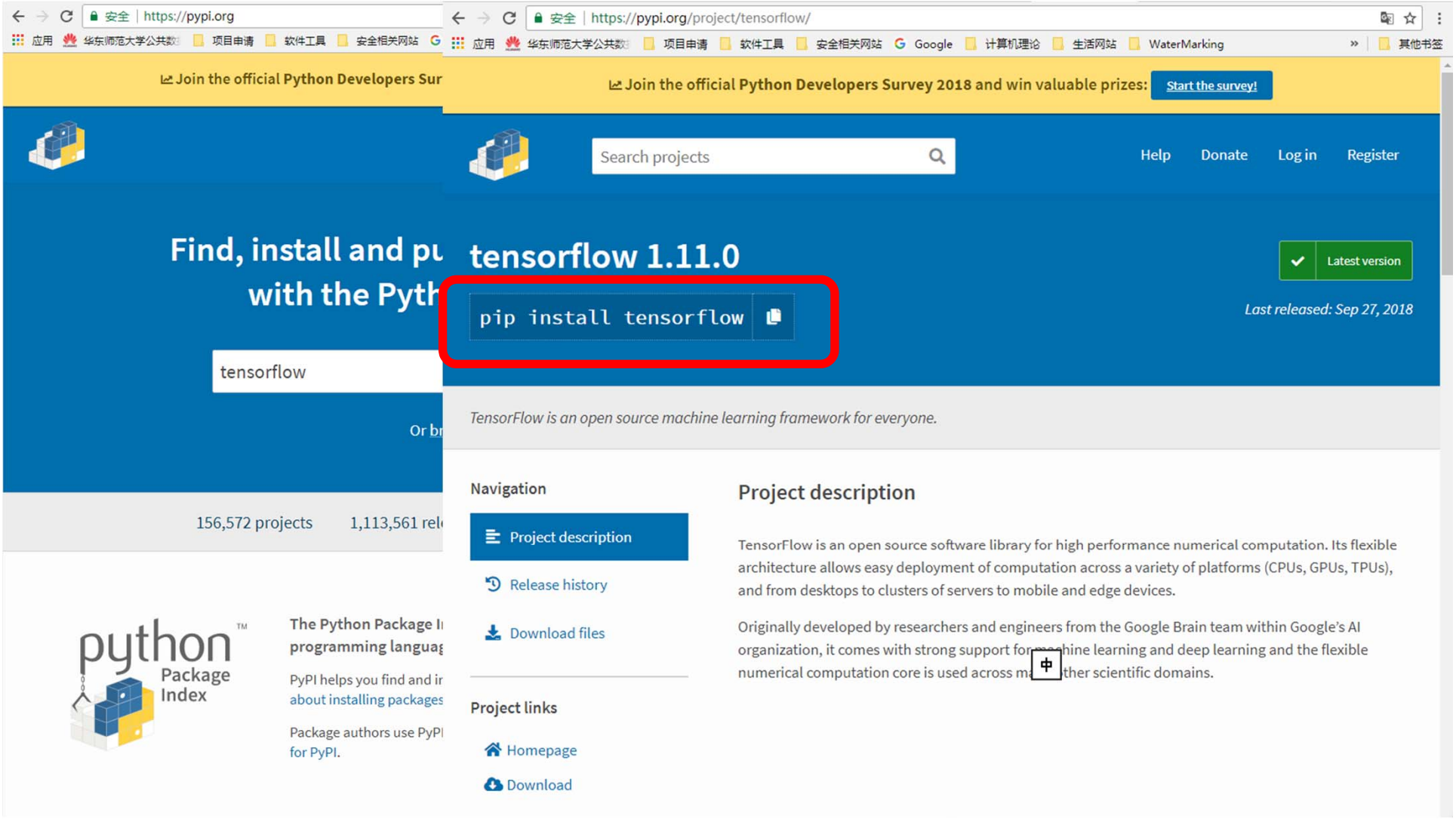
2. All built-in modules:

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

3. Third-party modules/packages, a package consists of several modules

   Manage third-party modules/package

   Look up at the website https://pypi.org/


安全 | https://pypi.org
Join the official Python Developers Sur
Find, install and pu
with the Pyth
tensorflow
Or br
156,572 projects
1,113,561 rel
python Package Index
The Python Package I
programming languag
PyPI helps you find and ir
about installing packages
Package authors use PyPI
for PyPI.
安全 | https://pypi.org/project/tensorflow/
应用 华东师范大学公共数 项目申请 软件工具 安全相关网站 Google 计算机理论 生活网站 WaterMarking 其他书签
Join the official Python Developers Survey 2018 and win valuable prizes: Start the survey!
Search projects
Help Donate Log in Register
tensorflow 1.11.0
pip install tensorflow
Latest version
Last released: Sep 27, 2018
TensorFlow is an open source machine learning framework for everyone.
Navigation
Project description
Release history
Download files
Project links
Homepage
Download
Project description
TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.
Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across ma ther scientific domains.
中

Manage third-parity modules/packages

Usage:

pip <command> [options] (in shell/cmd, not in python)

pip commands	description
pip download SomePackage[==version]	Download Some package, but not install
pip freeze [> requirements.txt]	Output installed packages in requirements format
pip list	list installed packages
pip install SomePackage[==version]	Install packages (online)
pip install SomePackage.whl	Install packages via whl files(offline)
pip install package1 package2 ...	Install package1、package2… (online)
pip install -r requirements.txt	Install packages list in requirements.txt file
pip install --upgrade SomePackage	Upgrade SomePackage
pip uninstall SomePackage[==version]	Uninstall SomePackage

Other install ways: setuptools , easy_install

Write a Module with main function

```
'''Fibonacci numbers module'''
def fib(n): # return Fibonacci series up to n
    result = [] # create an empty list
    a, b = 0, 1
    while a < n:
        result.append(a) # add a into the list
        a, b = b, a+b
    return result
if __name__ == "__main__":
import sys
lst = fib(int(sys.argv[1])) #get the first argument
print(lst)
```

Recap

- Class
  - The smallest class
  - Constructor __init__
  - Instance attributes and Class attributes
  - Access
  - Private and Public Attributes
  - Special method names
  - Modules