SI100B Introduction to Information Science and Technology
Python Programming

张海鹏 Haipeng Zhang
School of Information Science and Technology
ShanghaiTech University

Learning Objectives

- Exception handling
- File operations
- Modules

The assert statement

```
def Div(x,y):
    assert y!=0, "denominator is 0"
    return x/y

x = int(input("Input numerator:"))
y = int(input("Input denominator:"))
print(Div(x,y))
```

Input/Output

```
Input numerator:1
Input denominator:0
Traceback (most recent call last):
...
  File "C:\Users\Desktop\hello.py", line 2, in Div
    assert y!=0, "denominator is 0"
AssertionError: denominator is 0
```

Errors and Exceptions

There are (at least) two distinguishable kinds of errors

- Syntax errors: a.k.a. parsing errors, most common one

```
>>> while True print('Hello world')
File "<stdin>", line 1
while True print('Hello world')
^
SyntaxError: invalid syntax
```

- Exceptions: errors detected during execution

```
>>> 10 * (1/0)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Handling Exceptions

- Improve robustness and fault tolerance
- User-friendly error message
- Try statement

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops! That was no valid number.
Try again...")
```

If-Else vs. Exception Handling

- It is better to use exception handling than if-else
- Catch precise exception
- Proper exception handling for different exception

Exception hierarchy

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
           +-- ResourceWarning
```

The try statement

```
try_stmt  ::=  try1_stmt | try2_stmt

try1_stmt ::=  "try" ":" suite
               "finally" ":" suite

try2_stmt ::=  "try" ":" suite
               "except" [expression ["as" identifier]] ":" suite
                                .....
               "except" [expression ["as" identifier]] ":" suite
               ["else" ":" suite]
               ["finally" ":" suite]
```

The try statement

```
try:
    suite
finally:
    suite
```

- There is no exception handler
- But, the finally suite is always executed before leaving the try statement
- The finally clause is useful for releasing external resources (such as files or network connections), regardless of whether the use of the resource was successful

The try statement

```
try:  suite
except expression [as e1]:  suite
   ……
except [expression [as en]]:  suite
[else:   suite]
[finally:   suite]
```

- If there no finally clause, then it contains at least one except clause
- The last except can omit the expression, in this case, it will catch all the possible exception

The try statement

```
try:  suite
except expression [as e1]:  suite
   ……
except expression [as en]:  suite
[else:   suite]
[finally:   suite]
```

- If no exception occurs in the try clause, no exception handler is executed,
- **but** else clause is executed if no exception, no return, continue, or break statement was executed in try clause
- Exceptions in the else clause are not handled by the preceding except clauses

The try statement

```
try:  suite
except expression [as e1]:  suite
   ……
except expression [as en]:  suite
[else:   suite]
[finally:   suite]
```

When an exception occurs in the try suite, a search for an exception handler is started from the except clauses in turn until one is found that matches the exception (type-checking)

The try statement

```
try:  suite
except expression [as e1]:  suite
   ……
except expression [as en]:  suite
[else:   suite]
[finally:   suite]
```

If no except clause matches the exception in the current try block, the search continues in the surrounding code and on the invocation stack

The try statement

```
try:  suite
except expression [as e1]:  suite
   ……
except expression [as en]:  suite
[else:   suite]
[finally:   suite]
```

- When a matching except clause is found, the except clause's suite is executed
- When the end of this block is reached, execution continues normally after the entire try statement

The try statement

```
try:  suite
except expression [as e1]:  suite
   ……
except expression [as en]:  suite
[else:   suite]
[finally:   suite]
```

- If finally is present, it is always executed

The try statement

```
def foo(a,b):
    try:
        print("try-1")
        x = a/b
        print("try-2")
    except ZeroDivisionError:
        print("except")
        return
    else:
        print("else")
        return
    finally:
        print("finally")

foo(1,2)
```

Output

```
try-1
try-2
else
finally
>>>
```

If finally is present, it is always executed

The try statement

```
def foo(a,b):
    try:
        print("try-1")
        x = a/b
        print("try-2")
    except ZeroDivisionError:
        print("except")
        return
    else:
        print("else")
        return
    finally:
        print("finally")

foo(1,0)
print("after foo")
```

Raise here

Output

```
try-1
except
finally
after foo
>>>
```

If finally is present, it is always executed even if there is return

The try statement

```
def foo(a,b):
    try:
        print("try-1")
        x = a/b
        print("try-2")
    except AssertionError:
        print("except")
        return
    else:
        print("else")
        return
    finally:
        print("finally")

foo(1,0)
```

Output

```
try-1
finally
Traceback (most recent
call last):
  ...
ZeroDivisionError:
division by zero
>>>
```

The try statement

```
def foo(a,b):
    try:
        print("try-1")
        x = a/b
        print("try-2")
    except AssertionError:
        print("except-1")
        return
    finally:
        print("finally-1")
try:
    foo(1,0)
except ZeroDivisionError:
    print("except-2")
else:
    print("else-2")
finally:
    print("finally-2")
```

Output

```
try-1
finally-1
except-2
finally-2
>>>
```

Surrounding finally is also executed

The try statement

```
def foo(a,b):
    try:
        print("try-1")
        x = a/b
        print("try-2")
    except AssertionError:
        print("except-1")
        return
    finally:
        print("finally-1")
try:
    foo(1,0)
except IndexError:
    print("except-2")
else:
    print("else-2")
finally:
    print("finally-2")
```

Output

try-1
finally-1
finally-2
Traceback (most recent call last):
…
ZeroDivisionError: division by zero

Surrounding finally is also executed

Exception hierarchy

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
           +-- ResourceWarning
```

The try statement

```
x = [1,2,3]
try:
    print(x[0])
    print(x[3])
except IndexError:
    print("Out of range")

try:
    print(x[0])
    print(x[3])
except LookupError:
    print("Out of range")
```

Output

```
1
Out of range
1
Out of range
>>>
```

Can be more specific

Learning Objectives

- Exception handling
- File operations
- Modules

Open a file

- Before we can read the contents of the file we must tell Python which file we are going to work with and what we will be doing with the file
- This is done with the open() function
- open() returns a "file handle" - a variable used to perform operations on the file

Open()

- Syntax
  - file_handler_variable = open(filename, mode)

  - returns a handle use to manipulate the file
  - filename is a string (a string variable or a string constant)
  - mode is optional and should be 'r' if we are planning reading the file and 'w' if we are going to write to the file.

Open() modes

Character	Meaning
`'r'`	open for reading (default)
`'w'`	open for writing, truncating the file first
`'x'`	open for exclusive creation, failing if the file already exists
`'a'`	open for writing, appending to the end of the file if it exists
`'b'`	binary mode
`'t'`	text mode (default)
`'+'`	open a disk file for updating (reading and writing)

https://docs.python.org/3/library/functions.html#open

File handler as a sequence

- A file handle open for read can be treated as a sequence of strings where each line in the file is a string in the sequence
- We can use the for statement to iterate through a sequence
- Remember - a sequence is an ordered set

```
xfile = open('mbox.txt')
for line in xfile:
    print(line)
```

Read the 'whole' file

- We can **read** the whole file (newlines and all) into a single string.

```
>>> fh = open('mbox.txt')
>>> inp = fh.read()
>>> fh.close()
>>> print(len(inp))
94626
>>> print(inp[:20])
A text file (sometim
```

Read file into a list

- We can use **readlines()** to get a list.
- Each element in the list is a line.

```
>>> fh = open('mbox.txt')
>>> lines = fh.readlines()
>>> fh.close()
>>> print(len(lines))
4
>>> print(inp[:2])
['the first line', 'the second line']
```

Read using the with keyword

- A good practice - the file is properly closed after its suite finishes.

```
>>> with open('workfile') as f:
...     read_data = f.read()
>>> f.closed
True
```

File write

- The write() method writes any string to an open file.
- The write() method does not add a newline character ('\n') to the end of the string

```
>>> fh = open('test.txt', 'w')
>>> fh.write('Python is great\nI like Python')
>>> fh.close()
Python is great
I like Python
```

Other file operations

- Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.
- To use this module you need to import it first and then you can call any related functions.
  - import os
  - os.rename(current_file_name, new_file_name)
  - os.remove(file_name)
  - os.mkdir(newdir)
  - os.listdir(path)
  - …

Learning Objectives

- Exception handling
- File operations
- 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 before without copying its definition into the current program

Modules

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

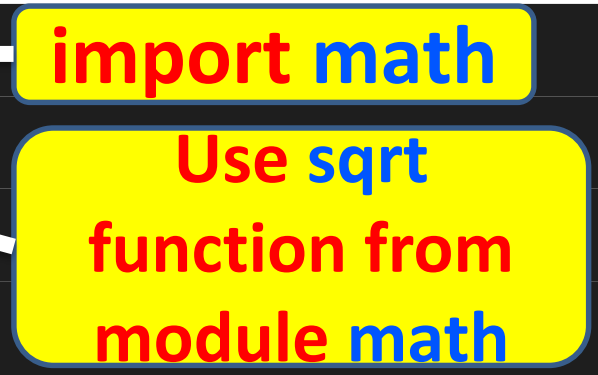
  ModuleName.py
- Within a module, the module's name (as a string) is available as the value of the global variable __name__
- To use a module,

  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
```



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)

```
File  Edit  Shell  Debug  Options  Window  Help
>>> help(math)
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module is always available.  It provides
    mathematical functions defined by the C standa

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians)

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radian

    atan2(y, x, /)
        Return the arc tangent (measured in radian

        Unlike atan(y/x), the signs of both x and

    atanh(x, /)
        Return the inverse hyperbolic tangent of x
```

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
    remainder(x, y, /)
        Difference between x and the closest integer multiple of y.

        Return x - n*y where n*y is the closest integer multiple of y.
        In the case where x is exactly halfway between two multiples of
        y, the nearest even value of n is used. The result is always exac

    sin(x, /)
        Return the sine of x (measured in radians).

    sinh(x, /)
        Return the hyperbolic sine of x.

    sqrt(x, /)
        Return the square root of x.

    tan(x, /)
        Return the tangent of x (measured in radians).

    tanh(x, /)
        Return the hyperbolic tangent of x.

    trunc(x, /)
        Truncates the Real x to the nearest Integral toward 0.

        Uses the __trunc__ magic method.

DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    (built-in)
```

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
```

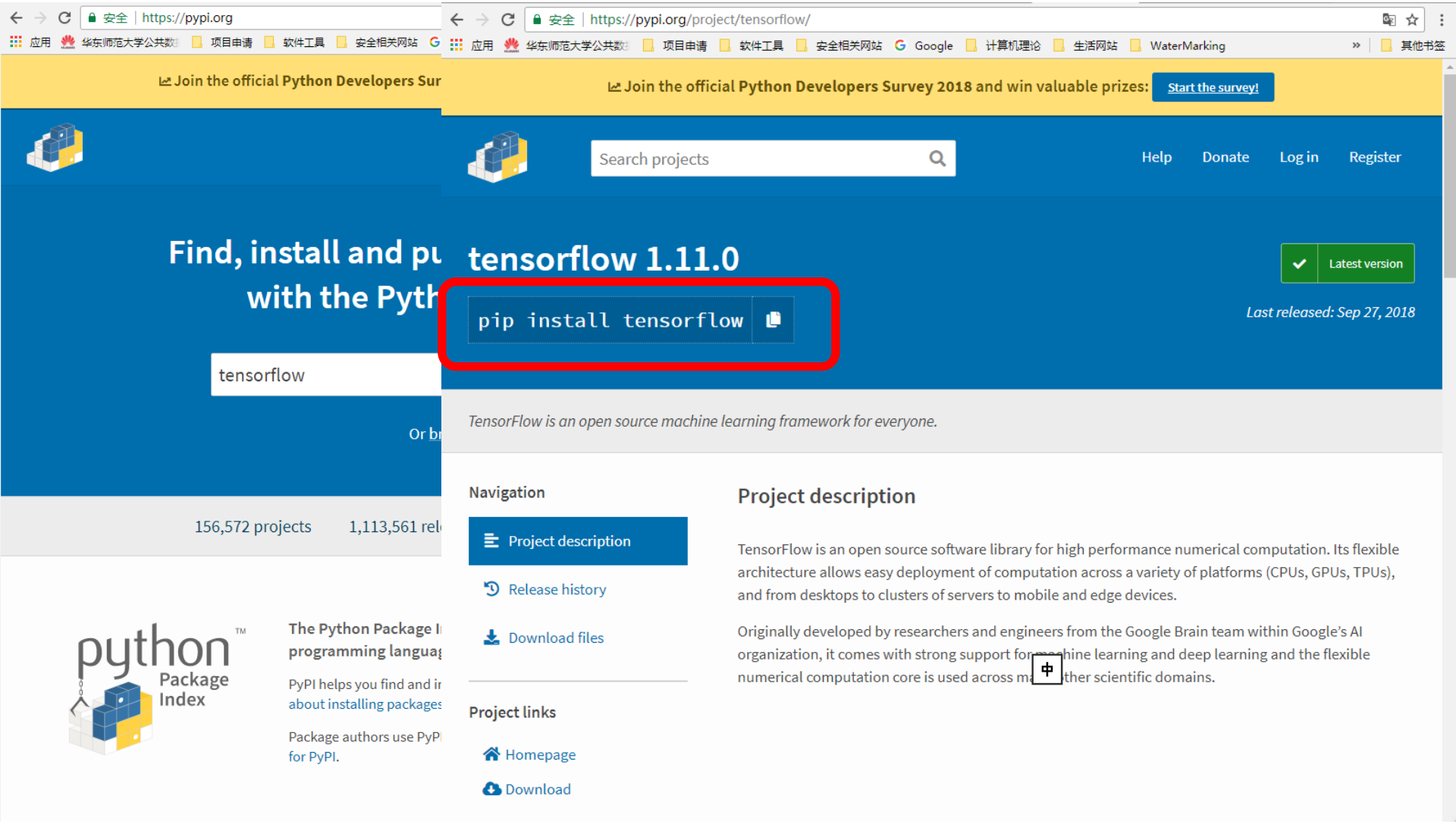
2. All built-in modules:

   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
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
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 PyP
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.

Manage third-parity modules/packages

Usage:

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

Find more packages at https://pypi.org/.

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

Modules

- Create our own module module_example.py

```
def func1(x):
    ...
def func2(x):
```

Modules

- import module:

  ```
  import module_example
  ```

- Use modules via "name space":

  ```
  >>> module_example.func1(1000)
  >>> module_example.__name__
  'module_example'
  ```

- can give it a local name:

  ```
  >>> fff = module_example.func1
  >>> fff(500)
  ```

Module search path

- When a module named module_example is imported, the interpreter first searches for a built-in module with that name.
- If not found, it searches for a file named module_example.py in a list of directories given by the variable sys.path. sys.path is initialized from these locations:
  - The directory containing the input script (or the current directory when no file is specified).
  - PYTHONPATH environment variable.
  - The installation-dependent default.

```
import sys

print(sys.path)
['C:\\Users\\HP', 'D:\\Anaconda3\\python37.zip', 'D:\\Anaconda3\\DLLs', 'D:\\Anaconda3\\lib',
'D:\\Anaconda3', '', 'D:\\Anaconda3\\lib\\site-packages', 'D:\\Anaconda3\\lib\\site-
packages\\win32', 'D:\\Anaconda3\\lib\\site-packages\\win32\\lib', 'D:\\Anaconda3\\lib\\site-
packages\\Pythonwin', 'D:\\Anaconda3\\lib\\site-packages\\IPython\\extensions',
'C:\\Users\\HP\\.ipython']
```

Readings (recommended)

- The Python Tutorial
  - 7. Input and Output
  - 8. Errors and Exceptions
  - 6. Modules

Recap

- Exception handling
- File operations
- Modules