Computational Physics

Chapter 1 Computer Science Basics

Weihua Gu

School of Physics & Astronomy

05-03-2020

The primary component of computational physics is to map physical problems into a computational framework.

In this chapter we aim to:

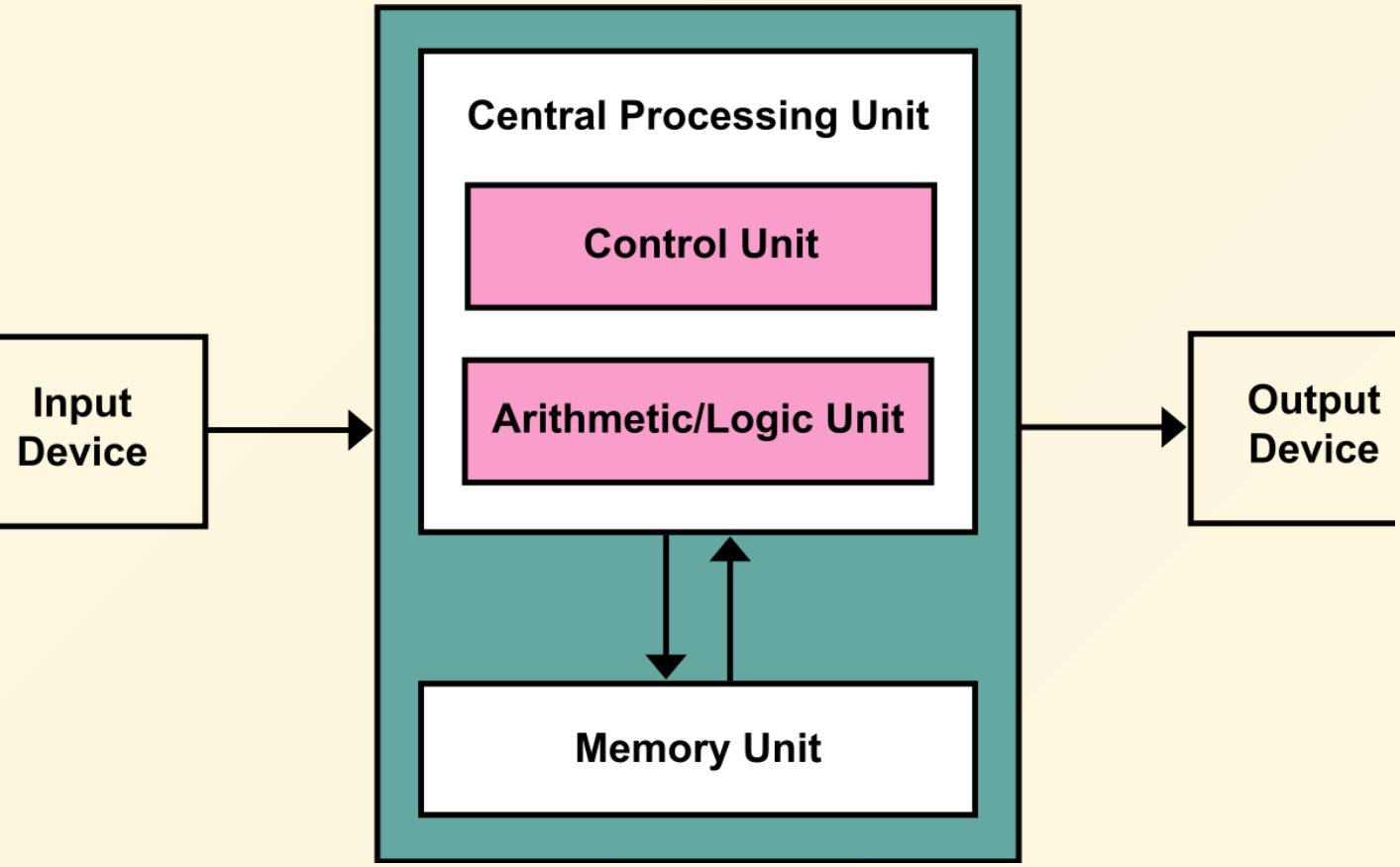
- understand the basic architecture of a simple computer,
- understand the main features of programming,
- get ready for Python programming.

> A computer does two things, and two things only: it performs calculations and it remembers the results of those calculations. But it does those two things extremely well. T

Guttag, Introduction to Computation and Programming Using Python_ With Application to Understanding Data, https://www.bilibili.com/video/av53706450/

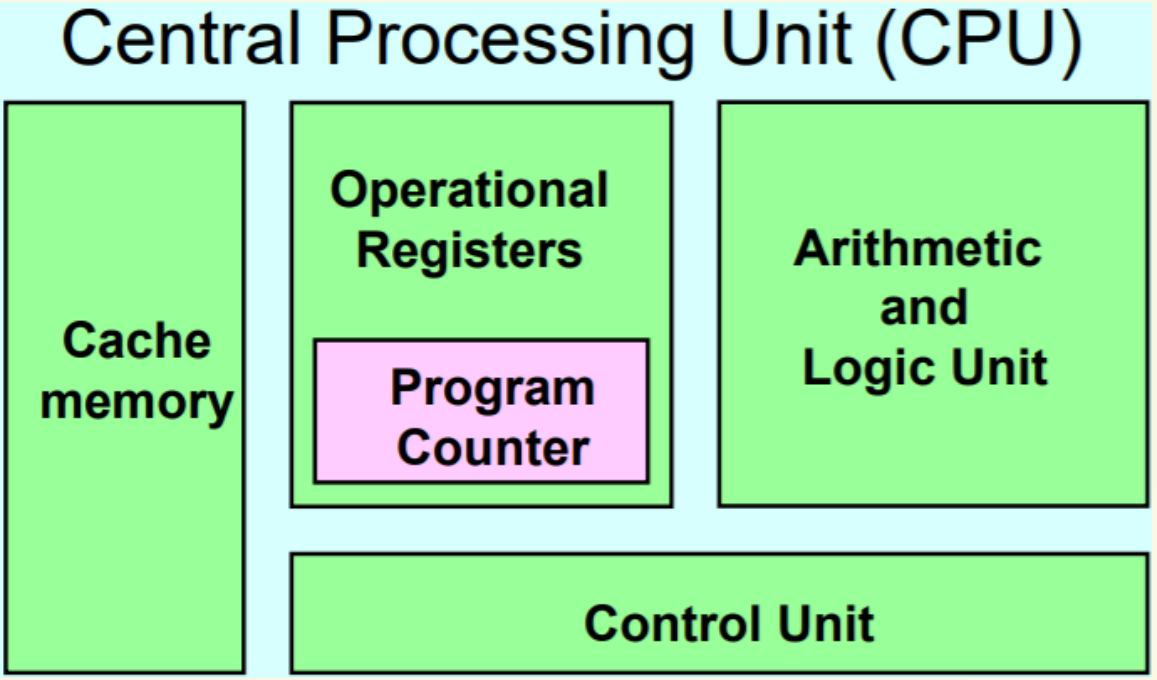
- hardware
- software

Computer Basics



Computer, handy, Pad, · · · primarily consist of three main parts (Von Newmann architecture):

- A processor (CPU)
- A main-memory system
- An I/O system



CPU

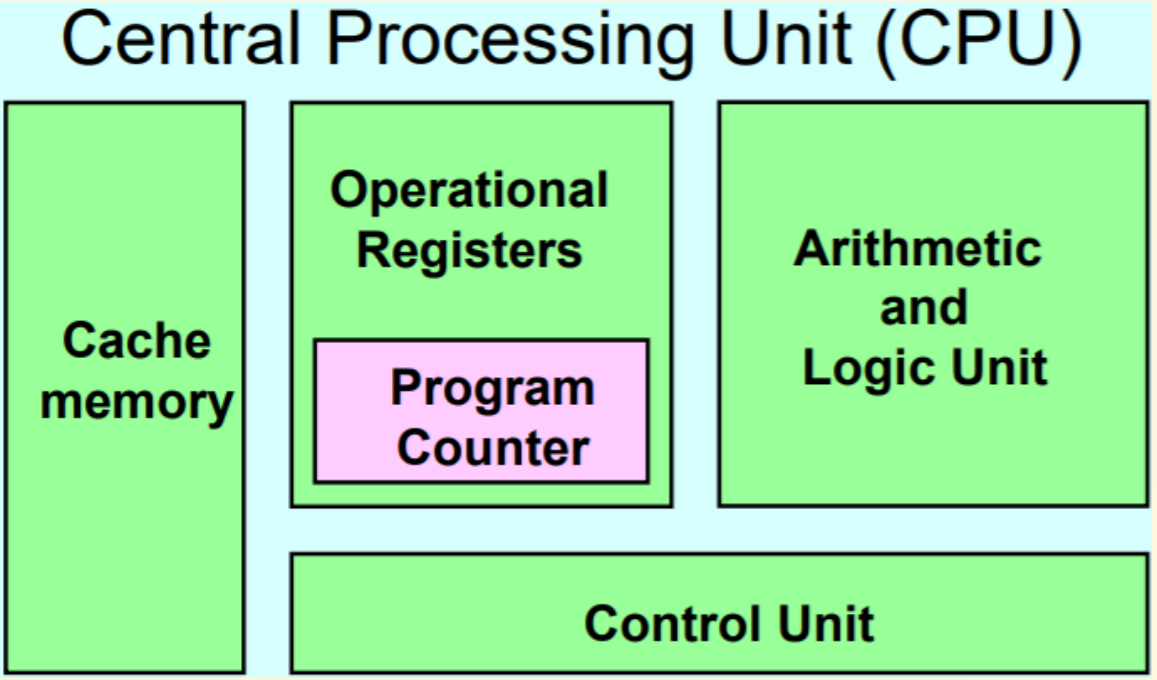
The CPU consists of :

- a control unit, registers,
- the arithmetic and logic unit(ALU),
- the instruction execution unit, and the interconnections among these components

Von Neumann bottleneck: the data transfer rate is smaller than the rate the CPU can work

ALU & Register

- Most computer operations are performed in the ALU
- For example, consider two numbers stored in the memory are to be added
  - They are brought into the processor, and the actual addition is carried out by the ALU. Then sum may be stored in the memory or retained in the processor for immediate use
- Typical arithmetic and logic operation
  - Addition, subtraction, multiplication, division, comparison, complement, etc.
- When operands are brought into the processor, they are stored in high-speed storage elements called registers.



Control unit

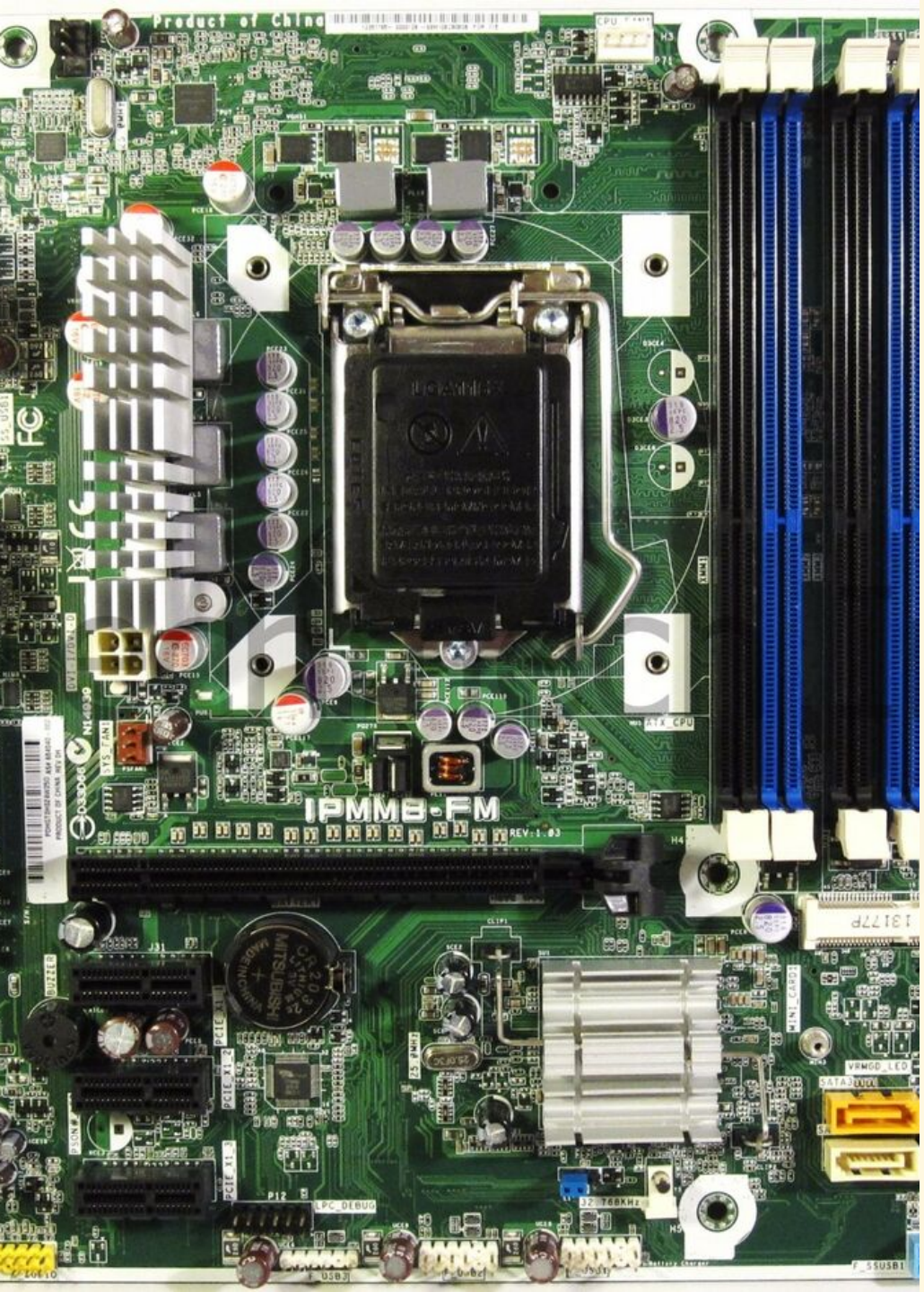
- Control unit serves as a coordinator of the memory, arithmetic and logic, and input/output units

Memory unit

- Memory unit is the storage area in which lists of instructions are kept when they are running and that contains the data needed by the running the instructions
- A computer has two classes of storage
  - primary memory (volatile memory: DRAM) and secondary memory (nonvolatile memory: magnetic disks)
  - volatile memory: storage that retains data only if it is receiving power
  - nonvolatile memory: storage that retains data even in the absence of a power source and that is used to store programs between runs, such as flash memory

Storage Cells & Word

- The memory consists of storage cells.
  - The storage cells are processed in groups of fixed size called **words**.
  - A word can contain a computer instruction, a storage address, or application data that is to be manipulated.
  - A distinct address is associated with each word location to provide easy access to any word in the memory.
- The number of bits in each word is often referred to as the word length of the computer
  - typical word length from 16 to 64 bits

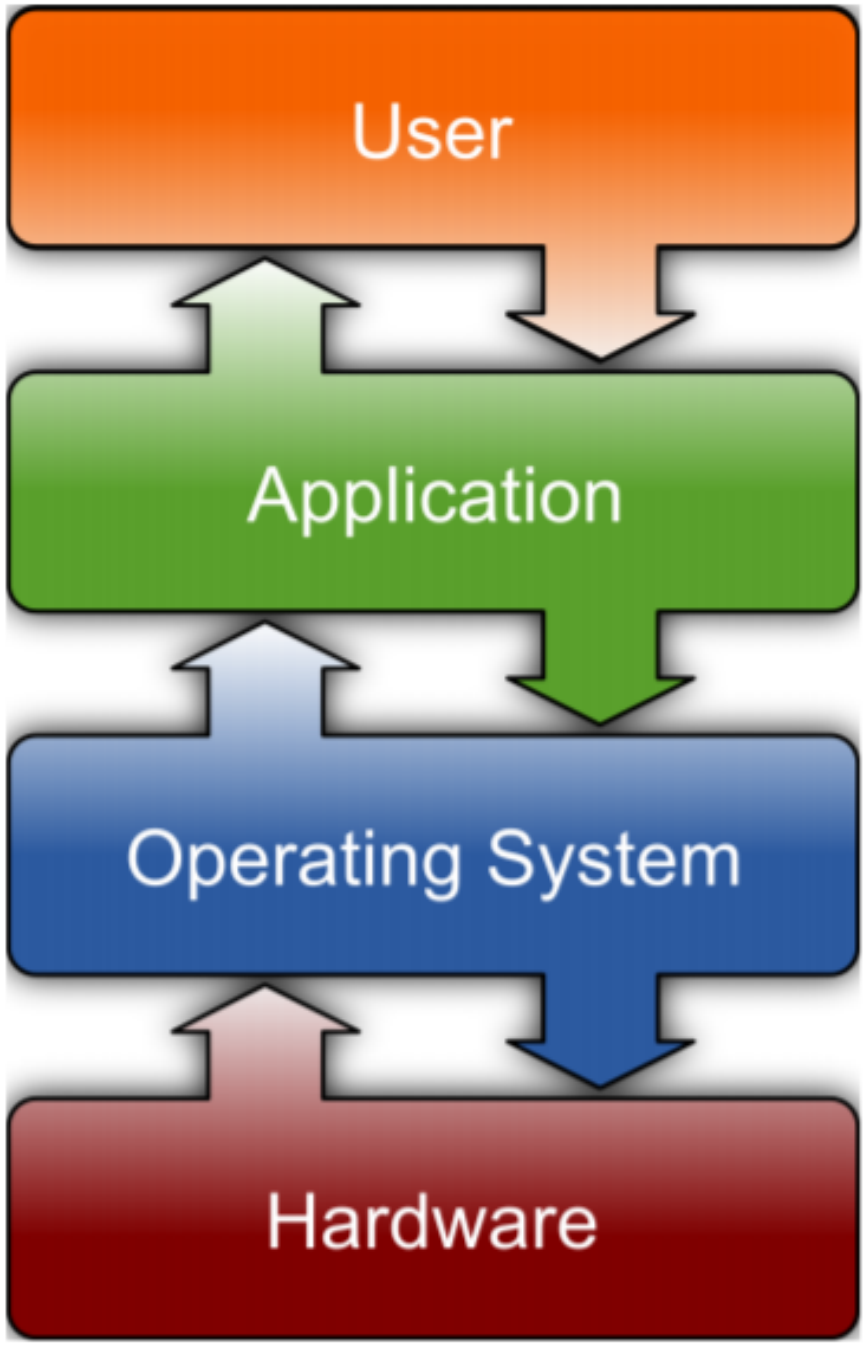


Additional parts of a computer

- mother board
- expansion cards
- power supply

Types of Computer

Super	parallel processing
Server	stores/accesses data.programs
workstation	expensive, specialized
PC	very common
microcontroller	tiny, specialized



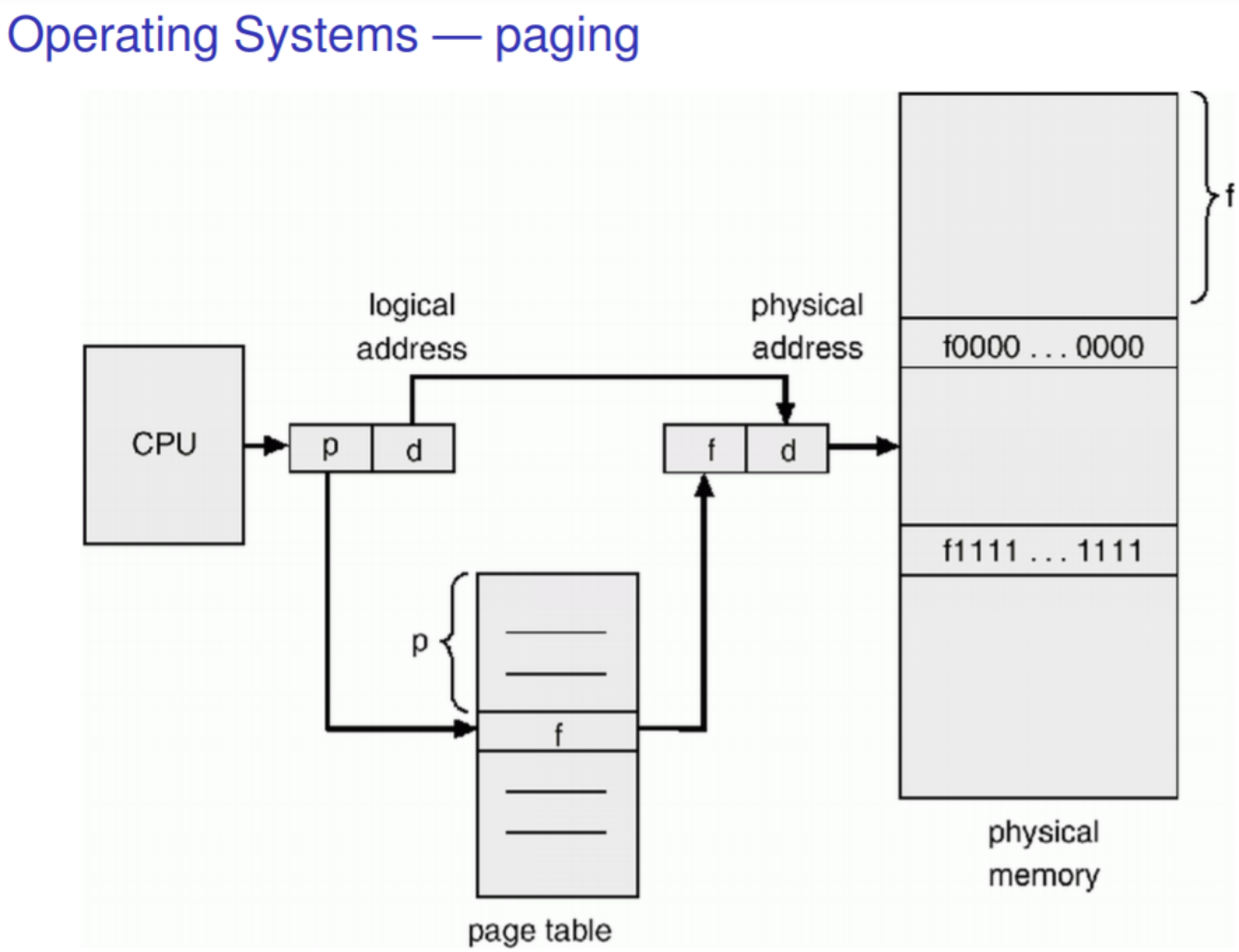
Interface of layers

Software aims to communicate with hardware

- operation system (OS): OS tells the computer what to do in an elementary way.
- application

Operation system (OS)

Different OSs: MS Windows, MacOS, Unix, Linux



> life is not simple for computers ...

Rubin Landau

Types of knowledge

- declarative Knowledge: composed of statements of fact

  > y is the square root of x if and only if $y * y = x$

- imperative knowledge: about how to accomplish something.

Imperative knowledge

- For example

> 1. Start with a guess, g
> 2. If $g * g$ is close enough to x, then g is a goodvapproximation of the square root of x
> 3. Otherwise, create a new guess by averaging g and x/g
> 4. Using this new guess, go back to step 2 ”

- Above description: a sequence of simple steps, together with a **flow of control** that specifies when each step is to be executed - **algorithm**

Algorithm

Defintion:

> an algorithm is a finite list of instructions that describe a computation that when executed on a set of inputs will proceed through a set of well-defined states and eventually produce an output.

- An algorithm is like a recipe.
- A good chef can make an unbounded number of tasty dishes by combining a few ingredients; given a small fixed set of primitive features a good programmer can produce an unbounded number of useful programs.

Some programming terms

- The difference between a *script* and a *program*

  > Well, a script is what you give the actors, but a program is what you give the audience.

- A *default value* is what will be used if nothing is specified.
- A *bug* is an error in a program.
- *Integrated development environment (IDE)*: a software application that provides comprehensive facilities for software development.
  - normally consists of at least a source code editor, build automation tools and a debugger.
  - some IDEs contain the necessary compiler, interpreter, or both

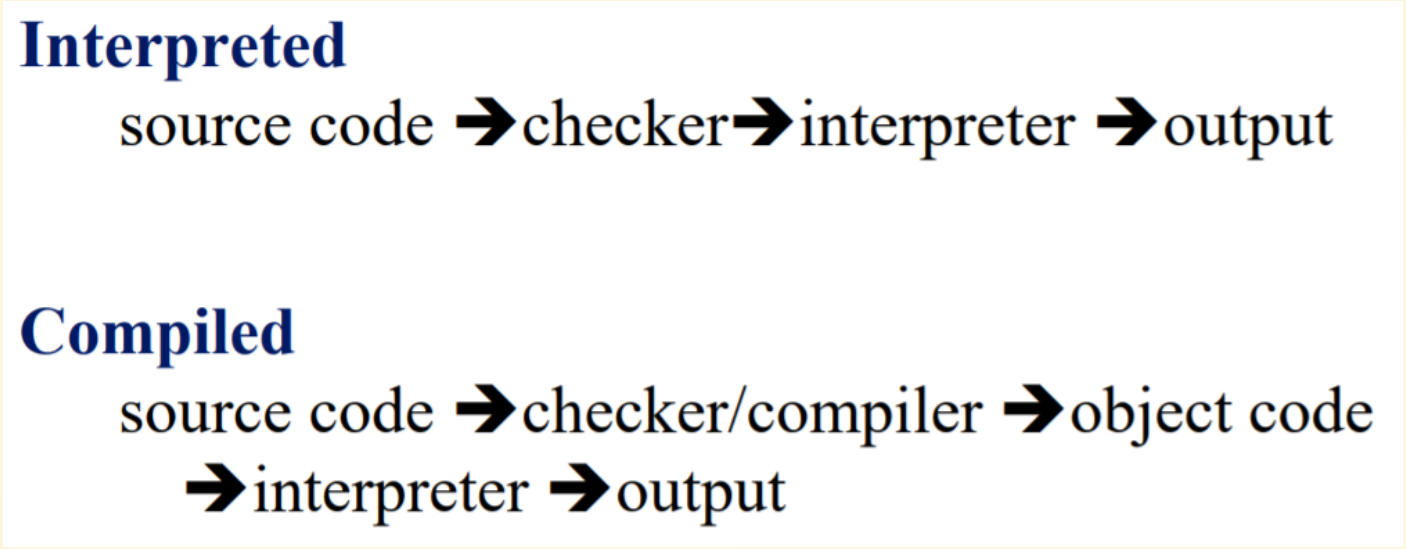
Programming language

- Programming language is the representation of algorithm.
- Different languages are better or worse for different kinds of applications.
  - MATLAB is a good language for manipulating vectors and matrices.
  - C is a good language for writing programs that control data networks.
  - PHP is a good language for building Web sites.
  - Python is a good general-purpose language.

Classification of programming languages

- Low-level versus high-level: whether using instructions and data objects at the level of the machine (e.g., move 64 bits of data from this location to that location) or using more abstract operations (e.g., pop up a menu on the screen).
- General versus targeted to an application domain: whether widely applicable or are finetuned to a domain.
  - SQL is designed to facilitate extracting information from relational databases, but you wouldn't want to use it build an operating system.

- Interpreted versus compiled:



  - It is often easier to debug programs written in languages that are designed to be interpreted.
  - Compiled languages usually produce programs that run more quickly and use less space.

What is a good program

Rubin Landau, Computational Physics, p.36

A good program should

- Give the correct answers.
- Be clear and easy to read, with the action of each part easy to analyze.
- Use trusted libraries.
- Be easy to use.
- Document itself for the sake of readers and the programmer.
- Be built up out of small programs that can be independently verified.
- Be easy to modify and robust enough to keep giving correct answers after modification and simple debugging.
- Document the data formats used.
- Be published or passed on to others to use and to develop further.

Modular programming

> Modular programming is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

https://en.wikipedia.org/wiki/Modular_programming

The benefits of using modular programming include:

- A single procedure can be developed for reuse, eliminating the need to retype the code many times.
- Programs can be designed more easily.
- Allowing many programmers to collaborate on the same application.
- The code is stored across multiple files.
- Code is short, simple and easy to understand.
- Errors can easily be identified, as they are localized to a subroutine or function.

- The same code can be used in many applications.
- The scoping of variables can easily be controlled.

https://www.techopedia.com/definition/25972/modular-programming

Object-oriented programming (OOP)

OOP is closely related with modular programming.

> Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data, in the form of fields (often known as **attributes** or properties), and code, in the form of procedures (often known as **methods**).

https://en.wikipedia.org/wiki/Object-oriented_programming

University.py

- can be used as a main script
- or as a module to be imported

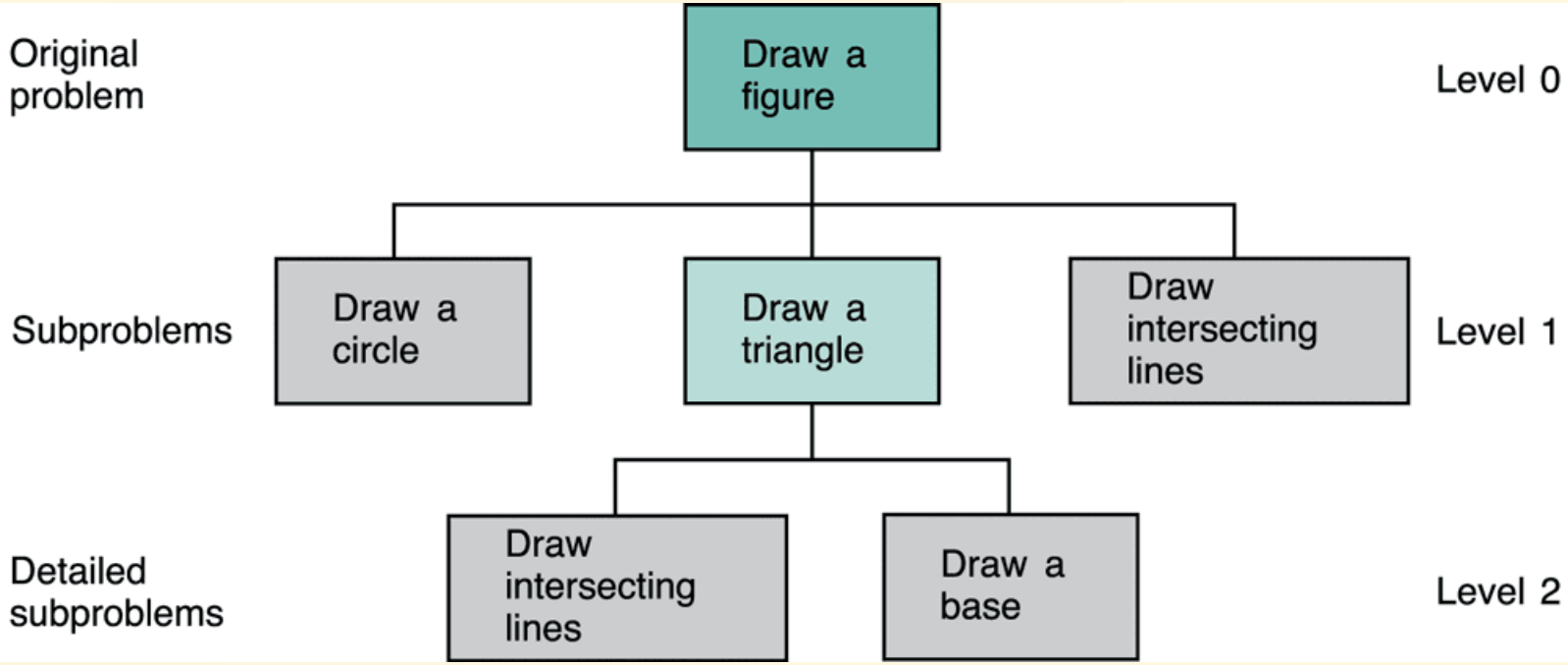
```
class University():         # University as an object
    def __init__(self, name='Shanghai Jiao Tong University', rank='#1'): # constructor of the object
        self.name = name    # attributes: name, rank
        self.rank = rank

    def get_name(self):     # method: get_name()
        print('My university name is {:s}.'.format(self.name))

if __name__ == '__main__':
    my_uni = University('Peking University') # create a Universtiy object
    print(my_uni.name)
    my_uni.get_name()
```

Top-down programming

- development outline first,
- add details for each of main steps
- add further refinement for more complex steps



Advantages of top-down design

- breaking the problem into parts helps us to clarify what needs to be done.
- at each step of refinement, the news parts become less complicated and, therefore, easier to figure out.
- parts of the solution may turn out to be reusable.
- breaking the problem into parts allows more than one person to work on the solution.

Bottom-up design

- start with the smallest solutions, the smallest subproblems,
- build up each solution until we reach the solution to the larger subproblems.
- similar to top-down design: we solve problem by combining the solutions to subproblems.
- different with top-down design: subproblmes are independent. Every sub(sub)problems are solved just once, their solutions are stored in a table and used for solving higher level subproblems.
- example: Fibonacci numbers computated by recursion.
- related with dynamic programming (vs. divide-and-conquer

Launch Python

Why Python?
Python provides a convenient and powerful scientific computing platform.

- Python is a high-level language. It includes commands for common operations in mathematical calculations, processing text, and manipulating files.
- Python can access many standard libraries, which are collections of programs that perform advanced functions like data visualization and image processing.

- Python comes with a command line interpreter, a program that executes Python commands as you type them. With Python, it is comparatively easy to quickly write, run, and debug programs.

Some Python programming related terms

- command line interpreter: IPython
  - IPython console
- IDE: `JupyterLab`, `Visual Studio Code`, `Spyder`, `PyCharm`
- libraries(modules): standard library & additional components
- packages: libraries + IPython + · · ·
- distribution (package collections): Anaconda, Enthought Canopy

Python packages

matplotlib	mathematics plotting library
mayavi	interactive and simplified 3D visualization
numpy	numerical python
pandas	Python data analysis library
python	standard library
seaborn	statistical data visualization
scipy	scientific python
pil	python imaging library
tkinter	graphical user interface (GUI) library

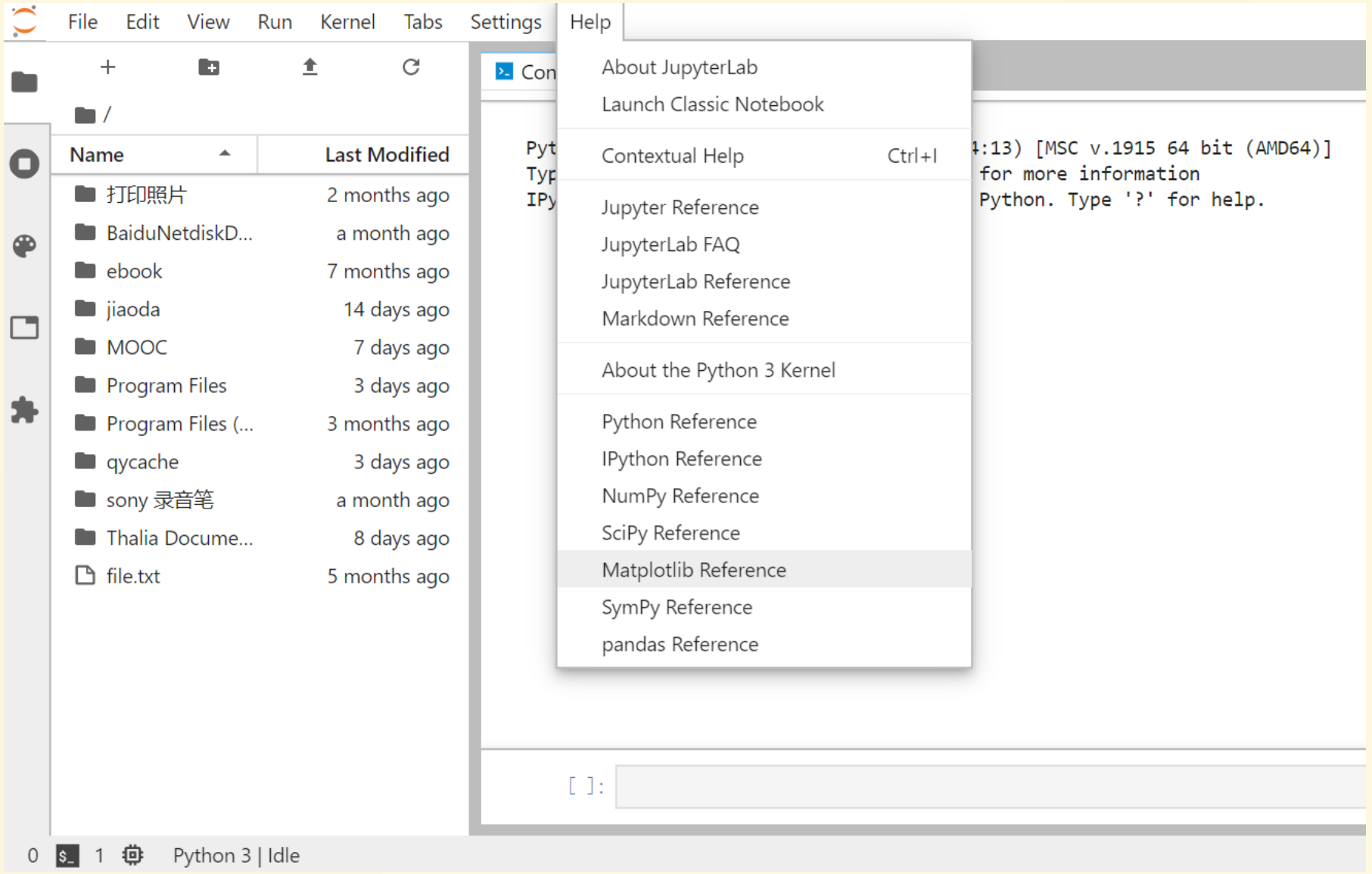
Import

To gain access to the libraries of useful functions and objects, you need to import them into your working environment.

```
import numpy as np
import pandas as pd
import matlablib.plot as plt
import seaborn as sns
import math
from tkinter import *
from os import open

np.sin(3)
math.pi
master = Tk() # from tkinter
fd = open('foo.txt')
```

Sources of help



Some common errors

- SyntaxError: the most common error for beginning programmers, usually means you typed a command incorrectly.

```
if q = 3:
  print('yes')
else:
  print('no')
```

- ImportError: cannot finnd a module. Most often, you type the name of the module or function incorrectly.

```
Import Numpy
```

Remember, Python names are case sensitive.

- AttributeError: an object does not possess an attribute or method. This is usually due to spelling the name of an attribute or method incorrectly.

```
np.cosine(3)
print(np.name)
```

- NameError: a variable, function, or module does not exist.

```
%reset
zlist
np.cos(3)
```

- IndexError: an index lies outside the range of a list or array.

```
x = [1, 2, 3]
x[3]
```

  Python's system of numbering elements starting at **0** rather than 1(`Matlab` does).

- IndexError: call a function with the wrong type of argument; or combine two objects in a way Python cannot interpret.

```
abs('-3')
x = [1, 2, 3]; x[1.5]
2 + x
```

Python in a Nutshell

- useful data structure
- Python I/O
- control structures

list of sample scripts:

1. numpy_array.py
2. io_console.py
3. read_file.py
4. read_web.py
5. save_load.py

Lists, tuples and `Numpy` arrays

Data structure: a bunch of numbers

- The most convenient Python data structure: the `Numpy` array,
- lists and tuples are also useful.
- creating a list

```
L = [1, 'a', max, 3 + 4j, 'Hello, world!']
```

- A tuple is a type of **immutable** object similar to a list. We can use tuple to set the size of a figure, or the size of an array.

```
t = ('keys', 3)
```

- create a one-dimensional `Numpy` array

```
import numpy as np
a = np.zeros(4)
```

- create a two-dimensional array

```
t = (2,3)
a = np.ones( t )
e = np.eye( t[1] )
b = np.array( [ [2, 3, 5], [7, 11, 13] ] )
```

Other methods to create Numpy arrays

```
min_value, max_value, step_size = 0, 10, 2
i = np.arange(min_value, max_value + step_size, step_size)

x_value = np.linspace(0, 1, 10)
y_value = np.logspace(0, 1, 10)
xy_value = np.array( [x_value, y_value] )

x = np.array([0, 1, 2])
y = np.array([0, 1])
X, Y = np.meshgrid(x, y)

r = np.random.random(t)
```

Indexing, slicing, reshaping

- array index starts from **0**, unlike `Matlab` starting from 1
- `:` : covering all, or delimiter for an integer range
- `-1` : the last index, or referring to backwards indexing

```
r[0, 1] == r[0][1]
r[-1]
r[0,-1]
r[::2,:]
r[::-1,1::2]
r1 = r.reshape( (3, 2) )  # r.shape = (2, 3)
less_than_onehalf = r[ r < 0.8 ]
```

Useful methods and attributes of Numpy arrays

```
print('np.size(r):\t{:d}'.format(np.size(r)))    # size is a method of np
# another way to determine the size of a numpy array
print('r.size:\t\t{:d}'.format(r.size))          # size is an attribute of a numpy array
str = ['rows', 'columns']

for i, value in enumerate(np.shape(r)):          # or: r.shape
    print('There are {1:d} {0:s} in the array.'.format(str[i], value))

# statistics: mean, sum, std, var
print('r.mean(0):\taveraging in column-wise\t', end='')
print(r.mean(0)) # return a numpy array, column-wise
print('r.mean(1):\taveraging in row wise\t\t', end='')
print(r.mean(1)) # return a numpy array, row-wise
print('r.mean():\taveraging over all data\t\t{:.5f}'.format(r.mean()))
```

Python I/O

- input data from *keyboard*
- output data to *screen*
- load data from a *file*
- save data to a *file*

Input data from *keyboard* & output data to *screen*

```
name = input('Please input your name: ')
age = eval(input('Please input your age: '))   # input an integer
height = float(input('Please input your height (in m):'))  # input a float value
print("My name is {0:s}. I'm {1:5.1f}m tall.\\".format(name, height))
print(r"I'm {:d} years old.\\".format(age))
```

```
Please input your name:  Shanghai Bund
Please input your age:  120
Please input your height (in m): 1200.95
My name is Shanghai Bund. I'm 1201.0m tall.\
I'm 120 years old.\\
```

format documentation and examples:
https://docs.python.org/3/library/string.html#formatspec

Load data from a *file*

- txt, csv, excel

1. We assign the variable name `data_set` to this array.
2. We must give `np.loadtxt` the le name as a string.
3. We must specify the *delimiter* as a string, in this case `'`.

```
home_dir = 'd:/jiaoda/ComputationalPhysics/CP2019-2020-2/'
data_dir = home_dir + 'SamplePrograms/chap1/'
data_set = np.loadtxt(data_dir + "HIVseries.csv", delimiter=',')
```

Direct import from the web

```
import requests as re
import numpy as np

web_file = re.get("http://www.physics.upenn.edu/biophys/" + \
                  "PMLS/Datasets/01HIVseries/HIVseries.csv")
web_file_text = web_file.text.split('\n')
temp_data = []
for line in web_file_text[: len(web_file_text) - 1]:
    x, y = line.split(',')
    temp_data += [ (float(x), float(y)) ]
data_set_web = np.array(temp_data)
```

Save data to a *file*

`NumPy` provides three convenient functions for saving data stored in arrays to a le:

- `np.save`: save a single array as a `NumPy` archive file (not human readable), with extension .npy
- `np.savez`: save multiple arrays as a single `NumPy` archive file (not human readable), with extension .npz
- `np.savetxt`: save a single array as a text file (human readable), with any extension

```
import numpy as np

x = np.linspace(0, 1, 1001)
y = 3*np.sin(x)**3 - np.sin(x)

np.save('x_values', x)
np.save('y_values', y)
np.savetxt('x_values.dat', x)  # readable
np.savetxt('y_values.dat', y)
np.savez('xy_values', x_vals=x, y_vals=y) # give the description name for the arrays

x2 = np.load('x_values.npy')
y2 = np.loadtxt('y_values.dat')
w = np.load('xy_values.npz')

print(x2 == x); print(y2 == y)
print(w['x_vals'] == x); print(w['y_vals'] == y)

for key in w.keys():
    print('The first value of w[{:s}] is {:.5f}.'.format(key, w[key][0]))
```

Control structures

- sequence
- selection
- iteration
- recursion

Selection

- Python uses `:` and **indented block** to indicate selection/loop block, unlike other languages need the keyword *end*.

1. `if`

```
if condition:
  block
```

2. `if-else`

```
if condition:
  block1
else:
  block2
```

3. `if-elif-..-else`

```
if d >= 0.0:
  print("Solutions are real")        # block 1
elif b == 0.0:
  print("Solutions are imaginary")   # block 2
else:
  print("Solutions are complex")     # block 3
print("Finished")
```

4. logical array

```
cubic_func = lambda x:  np.power(x[x > 5], 3)
cubic(n)
```

Use `if` to control the module

```
# in a python script
if __name__ == '__main__':
  block
```

- *_name_*: the title/name of the current module
- If the current module is directly employed as the main script, the following block shall be executed; otherwise the current module is called up by the main script, the following block shall be ignored.

Use `try` to deal with exceptions

1. `try-except`

```
# https://blog.csdn.net/qiqicos/article/details/79200089
try:
    a = 1 / 2
    print(a)
    print(m)  # 此处抛出python标准异常
    b = 1 / 0 # 此后的语句不执行
    print(b)
    c = 2 / 1
    print(c)
except NameError:
    print("Ops!!")
except ZeroDivisionError:
    print("Wrong math!!")
except:
    print("Error")
```

2. try-except-finally

```
try:
    a = 1 / 2
    print(a)
    print(m) # 抛出NameError异常
    b = 1 / 0
    print(b)
    c = 2 / 1
    print(c)
except:
    print("Error")
else:
    print("No error! yeah!")
finally:       # 是否异常都执行该代码块
    print("Successfully!")
```

3. `with-as`

```
# with expression [as variable]:
#     with-block
file = open("/tmp/foo.txt")
with open("/tmp/foo.txt") as file:
    data = file.read()
```

compare with:

```
try:
    data = file.read()
finally:
    file.close()
```

Iteration

1. `for`

```
import numpy as np
n = np.arange(0, 12, 2)
n_squared = [i**2 for i in n if i > 5]
```

2. `while`

```
a, b, c = 2, 2, -1
while (b**2 - 4*a*c >= 0):
  x = (-b + np.sqrt(b**2 - 4*a*c)) / (2*a)
  print("a = {:.4f}, x = {:.4f}".format(a, x))
  a = a - 0.3
print("done!")
```

Programming style

- Zen of Python
- PEP 8风格指南
- Python编程惯例
- 那些年我们踩过的那些坑

The Zen of Python, by Tim Peters (Python之禅)

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!