Tutorial for Python Basics

《<愚公移山> 列子・汤问篇》：编程的精髓

北山愚公者，年且九十，面山而居。　　惩山北之塞，出入之迂也，　　聚室而谋曰：

项目需求的诞生　　项目沟通

“吾与汝毕力平险，指通豫南，达于汉阴，可乎？”杂然相许。

项目目标

…… 叩石垦壤，箕畚运于渤海之尾。　　…… 遂率子孙荷担者三夫，

可以实现的技术方案　　一名工程管理人员和三名技术人员

……邻人京城氏之孀妻，有遗男，始龀，跳往助之。

一名力量较弱但满富工作激情的外协

河曲智叟笑而止之曰：“甚矣，汝之不惠！以残年余力，曾不能毁山之一毛，其如土石何？”北山愚公长息曰：

叙述编程实现

“ …… 虽我之死，有子存焉 ……

“IF”条件语句

子又生孙，孙又生子；子又有子，子又有孙；子子孙孙无穷匮也，　　而山不加增，何苦而不平？”

“while”循环语句　　“山平”循环结束条件

Python 基础

为什么学Python?

Guido van Rossum

编程语言排行榜TOP 50 榜单

排名	编程语言	流行度
1	**Python**	11.27%
2	C	11.16%
3	Java	10.46%
4	C++	7.50%

46 more rows

https://hellogithub.com › report › tiobe

2021年10月编程语言排行榜 - HelloGitHub

- 语法简单，极易上手
- 语法直观，极佳的可读性
- 丰富的库

  正则、文档生成、单元测试、线程、数据库、网页浏览器、CGI、FTP、电子邮件、XML、XML-RPC、HTML、WAV文件、密码系统、GUI .....

  只要你会Python，所有功能几乎都有相应程序接口。
- 完善的科学计算支持：PyTorch、TensorFlow、BrainPy等

Python 基础语法: 变量赋值

- Python 中的变量赋值不需要类型声明。
- 每个变量在内存中创建，都包括变量的标识，名称和数据这些信息。
- 每个变量在使用前都必须赋值，变量赋值以后该变量才会被创建。
- 等号 = 用来给变量赋值。
- 等号 = 运算符左边是一个变量名，等号 = 运算符右边是存储在变量中的值。例如：

```
In [6]: counter = 100 # 赋值整型变量
        miles = 1000.0 # 浮点型
        name = "John" # 字符串
```

```
In [1]: a = 10
        a + 20
Out[1]: 30

In [2]: a - 4
Out[2]: 6

In [3]: a * 2
Out[3]: 20

In [4]: a / 4
Out[4]: 2.5

In [5]: a ** 3
Out[5]: 1000
```

- Python 有一组关键字，这些关键字是保留字，不能用作变量名、函数名或任何其他标识符：

and	del	from	not	while	as	elif
global	or	with	assert	else	if	pass
yield	break	except	import	print	class	exec
in	raise	continue	finally	is	return	def
for	lambda	try				

- 允许使用字符、数字、下划线作为变量命名，数字不能在变量开头

```
76trombones = 'big parade'
```

```
  File "<ipython-input-6-ee59a172c534>", line 1
    76trombones = 'big parade'
    ^
SyntaxError: invalid syntax
```

```
more@ = 1000000
```

```
class = 'Advanced Theoretical Zymurgy'
```

Not Allowed

Python 基础语法: IF 条件语句

if语句用来做判断，并选择要执行的语句分支。
基本格式如下：

```
if CONDITION1:
    code_block(1)
elif CONDITION2:
    code_block(2)
elif CONDITION3:
    ...
...
else:
    code_block_else
```

其中elif是可选的，可以有任意多个，else是可选的，表示全都不满足条件时该执行的分支。

In [13]:
```
a = 4
if a > 3:
    print("hello world")
```

```
hello world
```

In [14]:
```
score = 77

if score >= 90:
    print("优秀")
elif 70 <= score < 90:
    print("良好")
elif 60 <= score < 70:
    print("及格")
else:
    print("不及格")
```

```
良好
```

Python 基础语法: 行和缩进

学习 Python 与其他语言最大的区别就是，Python 的代码块不使用大括号 {} 来控制类，函数以及其他逻辑判断。python 最具特色的就是用缩进来写模块。

In [1]:
```
if True:
    print ("True")
else:
    print ("False")
```

```
True
```

In [3]:
```
if True:
    print ("Answer")
    print ("True")
else:
    print ("Answer")
    # 没有严格缩进，在执行时会报错
  print ("False")
```

```
  File "<tokenize>", line 7
    print ("False")
    ^
IndentationError: unindent does not match any outer indentation level
```

Python 基础语法: for/while循环语句

```
while 判断条件(condition):
    执行语句(statements)……
```

In [5]:
```
n = 10

sum = 0
counter = 1
while counter <= n:
    sum = sum + counter
    counter += 1

print("1 到 %d 之和为: %d" % (n,sum))
```

```
1 到 10 之和为: 55
```

```
for iterating_var in sequence:
   statements(s)
```

In [3]:
```
n = 10

x = 0
for i in range(0, n+1):
    x += i
print("1 到 %d 之和为:  %d" % (n, x))
```

```
1 到 10 之和为:  55
```

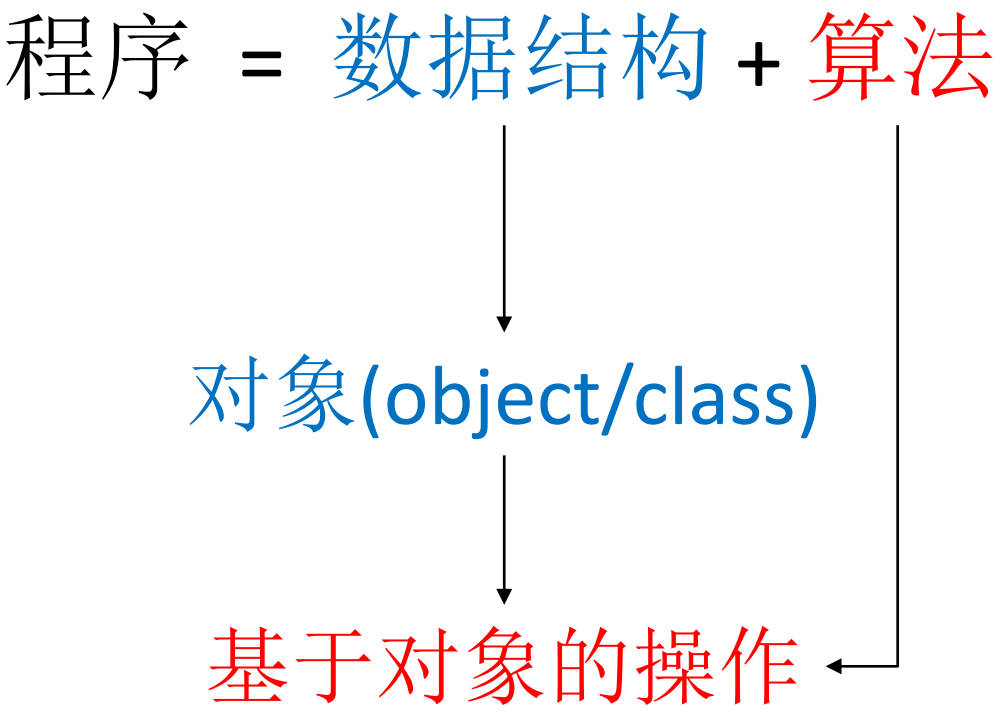
code

```
1  a = 1
2  while a < 7 :
3      if(a % 2 == 0):
4          print(a, "is even")
5      else:
6          print(a, "is odd")
7      a += 1
```

output

variables

程序是什么？



Python数据类型： 数字、字符串

Python有五个标准的数据类型：

- Numbers（数字）
- String（字符串）
- List（列表）
- Tuple（元组）
- Dictionary（字典）

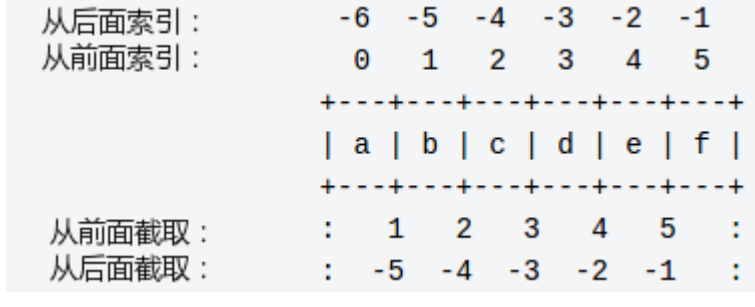
数字

```
In [1]: var1 = 1
        var2 = 10
```

字符串

```
In [2]: s = 'abcdef'

In [3]: s[1:5]
Out[3]: 'bcde'
```



Python数据类型： 列表

列表 (List)

- 列表用 [ ] 标识，是 python 最通用的复合数据类型。
- 列表中值的切割也可以用到变量 [头下标:尾下标]，就可以截取相应的列表，从左到右索引默认 0 开始，从右到左索引默认 -1 开始，下标可以为空表示取到头或尾。

```
In [7]: list = [ 'runoob', 786 , 2.23, 'john', 70.2 ]
        tinylist = [123, 'john']

In [8]: list                    # 输出完整列表
Out[8]: ['runoob', 786, 2.23, 'john', 70.2]

In [9]: list[0]                 # 输出列表的第一个元素
Out[9]: 'runoob'

In [10]: list[1:3]              # 输出第二个至第三个元素
Out[10]: [786, 2.23]
```

```
In [11]: list[2:]               # 输出从第三个开始至列表末尾的所有元素
Out[11]: [2.23, 'john', 70.2]

In [12]: tinylist * 2           # 输出列表两次
Out[12]: [123, 'john', 123, 'john']

In [13]: list + tinylist        # 打印组合的列表
Out[13]: ['runoob', 786, 2.23, 'john', 70.2, 123, 'john']
```

Python数据类型： 元组

元组 (tuple)

- 元组用 () 标识。内部元素用逗号隔开。但是元组不能二次赋值，相当于只读列表。

```
In [14]: tuple = ( 'runoob', 786 , 2.23, 'john', 70.2 )
         tinytuple = (123, 'john')

In [15]: tuple                    # 输出完整元组
Out[15]: ('runoob', 786, 2.23, 'john', 70.2)

In [16]: tuple[0]                 # 输出元组的第一个元素
Out[16]: 'runoob'

In [17]: tuple[1:3]               # 输出第二个至第四个（不包含）的元素
Out[17]: (786, 2.23)

In [18]: tuple[2:]                # 输出从第三个开始至列表末尾的所有元素
Out[18]: (2.23, 'john', 70.2)
```

```
In [19]: tinytuple * 2            # 输出元组两次
Out[19]: (123, 'john', 123, 'john')

In [20]: tuple + tinytuple        # 打印组合的元组
Out[20]: ('runoob', 786, 2.23, 'john', 70.2, 123, 'john')
```

"列表"和"元组"的不同。

```
In [21]: tuple[2] = 1000          # 元组中是非法应用
---------------------------------------------------------------
TypeError                                 Traceback (most r
<ipython-input-21-2a8fa2216ea0> in <module>
----> 1 tuple[2] = 1000          # 元组中是非法应用

TypeError: 'tuple' object does not support item assignment

In [22]: list[2] = 1000           # 列表中是合法应用
```

Python数据类型： 字典

字典 (dict)

- 字典(dictionary)是除列表以外python之中最灵活的内置数据结构类型。列表是有序的对象集合，字典是无序的对象集合。
- 两者之间的区别在于：字典当中的元素是通过键来存取的，而不是通过偏移存取。
- 字典用{ }标识。字典由索引(key)和它对应的值value组成。

```
In [23]: dict = {}
         dict['one'] = "This is one"
         dict[2] = "This is two"

In [24]: tinydict = {'name': 'runoob','code':6734, 'dept': 'sales'}

In [26]: dict['one']          # 输出键为'one' 的值
Out[26]: 'This is one'

In [27]: dict[2]              # 输出键为 2 的值
Out[27]: 'This is two'
```

```
In [28]: tinydict             # 输出完整的字典
Out[28]: {'name': 'runoob', 'code': 6734, 'dept': 'sales'}

In [29]: tinydict.keys()      # 输出所有键
Out[29]: dict_keys(['name', 'code', 'dept'])

In [30]: tinydict.values()    # 输出所有值
Out[30]: dict_values(['runoob', 6734, 'sales'])
```

Implement HH derivatives

$$\frac{dm}{dt} = \alpha_m(1-m) - \beta_m$$

$$\frac{dh}{dt} = \alpha_h(1-h) - \beta_h$$

$$\frac{dn}{dt} = \alpha_n(1-n) - \beta_n$$

```python
import math

num = 3  # 神经元数目
I = 5  # 外界电流大小
Vs, ms, ns, hs = [-10, 10, 20], [0]*num, [0]*num, [0]*num

for i in range(num):
    V, m, n, h = Vs[i], ms[i], ns[i], hs[i]

    alpha = 0.1 * (V + 40) / (1 - math.exp(-(V + 40) / 10))
    beta = 4.0 * math.exp(-(V + 65) / 18)
    dmdt = alpha * (1 - m) - beta * m

    alpha = 0.07 * math.exp(-(V + 65) / 20.)
    beta = 1 / (1 + math.exp(-(V + 35) / 10))
    dhdt = alpha * (1 - h) - beta * h

    alpha = 0.01 * (V + 55) / (1 - math.exp(-(V + 55) / 10))
    beta = 0.125 * math.exp(-(V + 65) / 80)
    dndt = alpha * (1 - n) - beta * n

    I_Na = (120. * m ** 3 * h) * (V - 50.)
    I_K = (36. * n ** 4) * (V - -77)
    I_leak = 0.03 * (V - -54.)
    dVdt = (- I_Na - I_K - I_leak + I) / 1.

    print(f'Neuron {i}: dmdt={dmdt}, dndt={dndt}, dhdt={dhdt}, dVdt={dVdt}')
```

$$C\frac{dV}{dt} = -(\bar{g}_{Na}m^3h(V-E_{Na}) + \bar{g}_K n^4(V-E_K) + g_{leak}(V-E_{leak})) + I(t)$$

```
Neuron 0: dmdt=3.157187089473768, dndt=0.4550552067161185, dhdt=0.0044749502844695305, dVdt=3.68
Neuron 1: dmdt=5.033918274531521, dndt=0.65097870690175, dhdt=0.0016462422099206377, dVdt=3.08
Neuron 2: dmdt=6.014909469941068, dndt=0.7504150428313138, dhdt=0.00099849637362994 8, dVdt=2.7800
```

Python语法 ： 函数定义

如何重复利用代码逻辑？

你可以定义一个由自己想要功能的函数，以下是简单的规则：

- 函数代码块以 **def** 关键词开头，后接函数标识符名称和圆括号 ()。
- 任何传入参数和自变量必须放在圆括号中间，圆括号之间可以用于定义参数。
- 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。
- 函数内容以冒号起始，并且缩进。
- return [表达式] 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回None。

```
def 函数名（参数列表）：
    函数体
```

In [9]:

```
# 计算面积函数
def area(width, height):
    return width * height

w = 4
h = 5
print("width =", w, " height =", h, " area =", area(w, h))
```

```
width = 4  height = 5  area = 20
```

必需参数

必需参数须以正确的顺序传入函数。调用时的数量必须和声明时的一样。

In [10]:
```
#可写函数说明
def printme( str ):
   "打印任何传入的字符串"
   print (str)

# 调用 printme 函数，不加参数会报错
printme()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent c
<ipython-input-10-8260cb4a155b> in <module>
      5
      6 # 调用 printme 函数，不加参数会报错
----> 7 printme()

TypeError: printme() missing 1 required positional argument: 'str'
```

关键字参数

关键字参数和函数调用关系紧密，函数调用使用关键字参数来确定传入的参数值。

使用关键字参数允许函数调用时参数的顺序与声明时不一致，因为 Python 解释器能够用参数名匹配参数值。

In [11]:
```
#可写函数说明
def printinfo( name, age ):
   "打印任何传入的字符串"
   print ("名字: ", name)
   print ("年龄: ", age)

#调用printinfo函数
printinfo( age=50, name="runoob" )
```

```
名字:  runoob
年龄:  50
```

默认参数

调用函数时，如果没有传递参数，则会使用默认参数。以下实例中如果没有传入 age 参数，则使用默认值：

In [12]:
```
#可写函数说明
def printinfo( name, age = 35 ):
   "打印任何传入的字符串"
   print ("名字: ", name)
   print ("年龄: ", age)

#调用printinfo函数
printinfo( age=50, name="runoob" )
print ("------------------------")
printinfo( name="runoob" )
```

```
名字:  runoob
年龄:  50
------------------------
名字:  runoob
年龄:  35
```

不定长参数

你可能需要一个函数能处理比当初声明时更多的参数。这些参数叫做不定长参数，和上述 2 种参数不同，声明时不会命名。

In [13]:
```
# 可写函数说明
def printinfo( arg1, *vartuple ):
   "打印任何传入的参数"
   print ("输出: ")
   print (arg1)
   print (vartuple)

# 调用printinfo 函数
printinfo( 70, 60, 50 )
```

```
输出:
70
(60, 50)
```

```
# 可写函数说明
def printinfo( arg1, **vardict ):
   "打印任何传入的参数"
   print ("输出: ")
   print (arg1)
   print (vardict)

# 调用printinfo 函数
printinfo(1, a=2,b=3)
```

```
输出:
1
{'a': 2, 'b': 3}
```

```python
def hh_derivative(V, m, h, n, I):
    alpha = 0.1 * (V + 40) / (1 - math.exp(-(V + 40) / 10))
    beta = 4.0 * math.exp(-(V + 65) / 18)
    dmdt = alpha * (1 - m) - beta * m

    alpha = 0.07 * math.exp(-(V + 65) / 20.)
    beta = 1 / (1 + math.exp(-(V + 35) / 10))
    dhdt = alpha * (1 - h) - beta * h

    alpha = 0.01 * (V + 55) / (1 - math.exp(-(V + 55) / 10))
    beta = 0.125 * math.exp(-(V + 65) / 80)
    dndt = alpha * (1 - n) - beta * n

    I_Na = (120. * m ** 3 * h) * (V - 50.)
    I_K = (36. * n ** 4) * (V - -77)
    I_leak = 0.03  * (V - -54.)
    dVdt = (- I_Na - I_K - I_leak + I) / 1.

    print(f'Neuron {i}: dmdt={dmdt}, dndt={dndt}, dhdt={dhdt}, dVdt={dVdt}')
```

```python
num = 3  # 神经元数目
I = 5  # 外界电流大小
Vs, ms, ns, hs = [-10, 10, 20], [0]*num, [0]*num, [0]*num

for i in range(num):
    V, m, n, h = Vs[i], ms[i], ns[i], hs[i]
    hh_derivative(V, m, n, h, I)
```

```
Neuron 0: dmdt=3.157187089473768, dndt=0.4550552067161185, dhdt=0.0044749502844695305, dVdt=3.68
Neuron 1: dmdt=5.033918274531521, dndt=0.65097870690175, dhdt=0.0016462422099206377, dVdt=3.08
Neuron 2: dmdt=6.014909469941068, dndt=0.7504150428313138, dhdt=0.000998496373629948, dVdt=2.780
```

Python语法 ： 类/对象

class: Python一切皆对象

在python中，一切皆对象。数字、字符串、元组、列表、字典、函数、方法、类、模块等等都是对象，包括你的代码。

```
In [1]: a = -1
        id(a)
Out[1]: 140736164144928
```

Python 的所有对象都有三个特性：

- **身份**：每个对象都有一个唯一的身份标识自己，任何对象的身份都可以使用内建函数 id() 来得到，可以简单的认为这个值是该对象的内存地址。

- **类型**：对象的类型决定了对象可以保存什么类型的值，有哪些属性和方法，可以进行哪些操作，遵循怎样的规则。可以使用内建函数 type() 来查看对象的类型。

```
In [2]: type(a)
Out[2]: int

In [3]: type(type(a))
Out[3]: type
```

- **属性和方法**：大部分 Python 对象有属性、值或方法，使用句点（.）标记法来访问属性。最常见的属性是函数和方法，一些 Python 对象也有数据属性，如：类、模块、文件等。

```
In [5]: a.real
Out[5]: 1

In [8]: a.bit_length()
Out[8]: 1

In [10]: a.__abs__()
Out[10]: 1
```

如何创建类？

```python
class Employee(object):  # 类

    empCount = 0  # 类变量

    def __init__(self, name, salary):
        # self 代表类的实例，self 在定义类的方法时是必须有的，
        # 虽然在调用时不必传入相应的参数。

        # 属性
        self.name = name
        self.salary = salary

        Employee.empCount += 1

    # 方法
    def displayCount(self):
        print("Total Employee %d" % Employee.empCount)

    def displayEmployee(self):
        print("Name : ", self.name,  ", Salary: ", self.salary)
```

如何使用类？

初始化

```
In [4]: em = Employee(name='test', salary=1000)

In [3]: a = 'this is a string'

        str.__init__
Out[3]: <slot wrapper '__init__' of 'object' objects>

In [5]: b = 10

        int.__init__
Out[5]: <slot wrapper '__init__' of 'object' objects>
```

属性与方法

```
In [14]: b + 1
Out[14]: 11

In [15]: b.__add__(1)
Out[15]: 11

In [16]: b - 1
Out[16]: 9

In [17]: b.__sub__(1)
Out[17]: 9
```

实现我们自己的加减乘除

In [2]:
```
class Area:
    def __init__(self, h, w):
        self.h = h
        self.w = w

    def __add__(self, oc):
        return Area(self.h + oc.h, self.w + oc.w)

    def __truediv__(self, oc):
        return Area(self.h / oc.h, self.w / oc.w)

    def __mul__(self, oc):
        return Area(self.h * oc.h, self.w * oc.w)

    def __sub__(self, oc):
        return Area(self.h - oc.h, self.w - oc.w)

    def __repr__(self):
        return 'Area({}, {})'.format(self.h, self.w)

    def size(self):
        return self.h * self.w
```

In [3]:
```
c1 = Area(10, 20)
c2 = Area(5, 4)
```

In [4]:
```
c1 + c2
```

Out[4]: Area(15, 24)

In [5]:
```
c1 - c2
```

Out[5]: Area(5, 16)

In [6]:
```
c1 / c2
```

Out[6]: Area(2.0, 5.0)

In [7]:
```
c1 * c2
```

Out[7]: Area(50, 80)

In [8]:
```
c1.size()
```

Out[8]: (200, 20)

Python语法 ： 模块使用

在计算机程序的开发过程中，随着程序代码越写越多，在一个文件里代码就会越来越长，越来越不容易维护。

为了编写可维护的代码，我们把很多函数分组，分别放到不同的文件里，这样，每个文件包含的代码就相对较少，很多编程语言都采用这种组织代码的方式。在Python中，一个.py文件就称之为一个模块（Module）。

```
import math

type(math)
```

```
module
```

```
# 模块定义好后，我们可以使用 import 语句来引入模块，语法如下：

import module1[, module2[,... moduleN]]
```

```
# Python 的 from 语句让你从模块中导入一个指定的部分到当前命名空间中。语法如下：

from modname import name1[, name2[, ... nameN]]
```

```
# 把一个模块的所有内容全都导入到当前的命名空间也是可行的，只需使用如下声明：

from modname import *
```

```
In [15]: import math

print('The sqrt of 16 = ', math.sqrt(16))
```

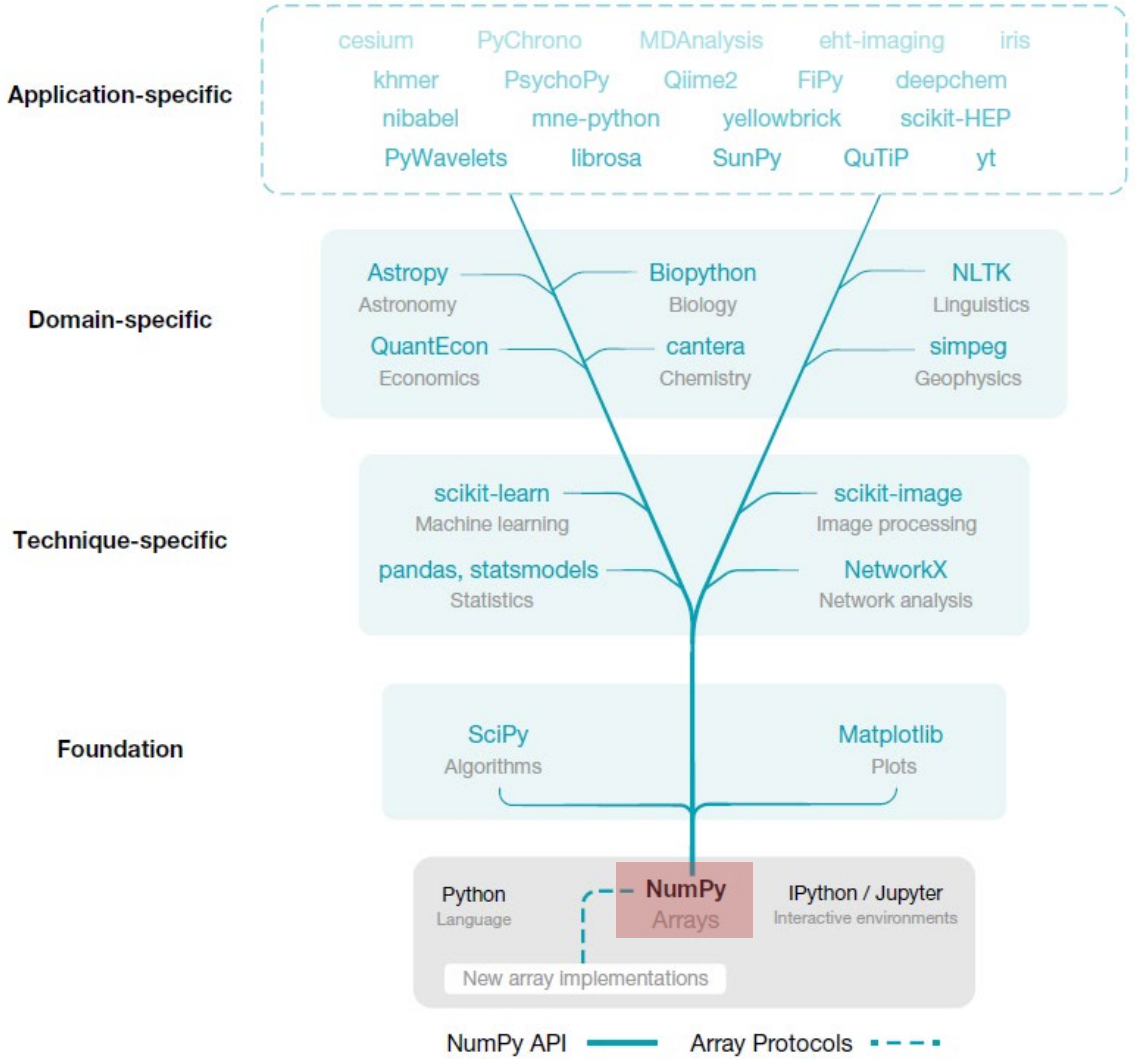
```
The sqrt of 16 =  4.0
```

```
In [16]: from math import sqrt

print('The sqrt of 16 = ', sqrt(16))
```

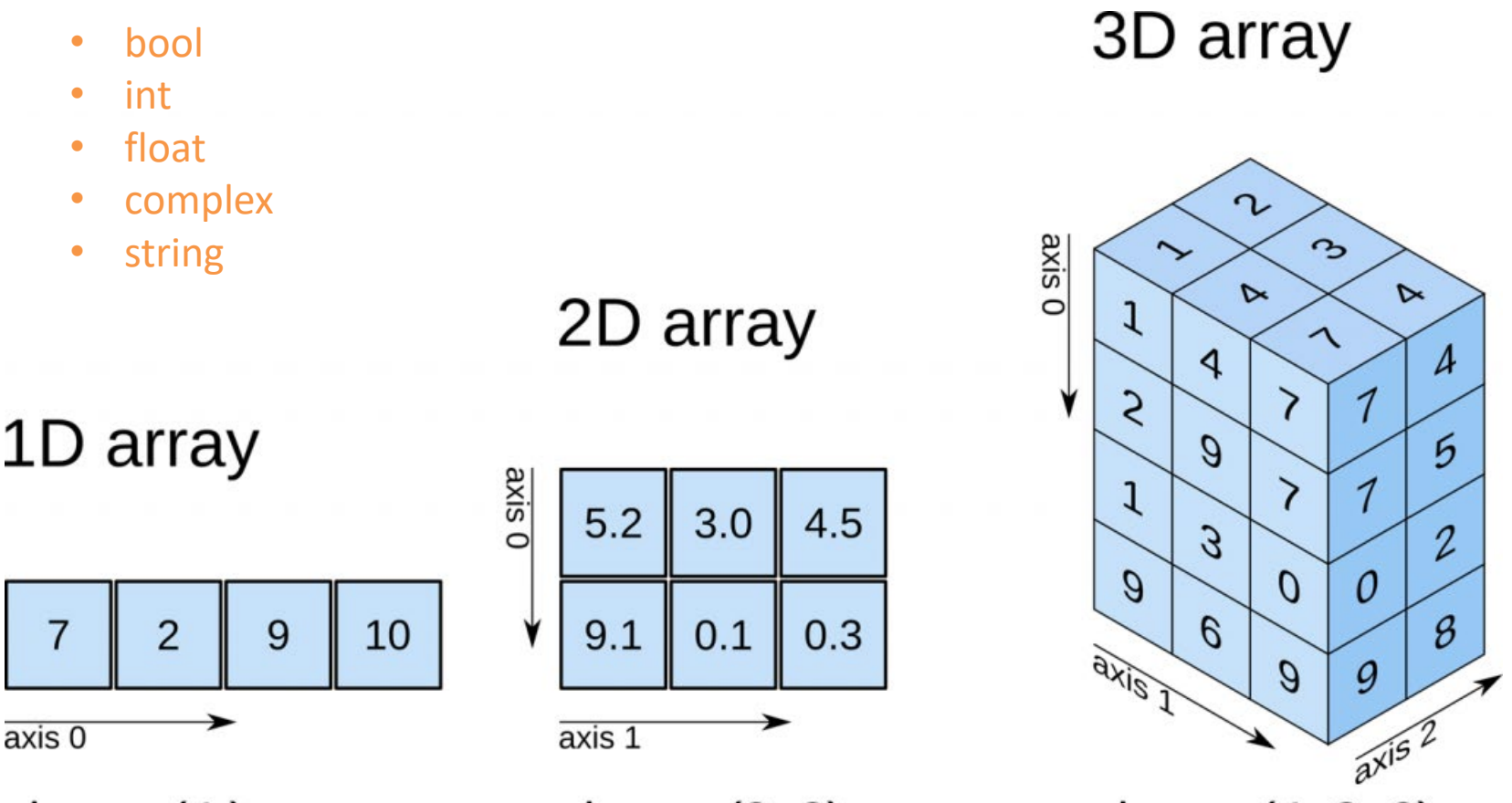
```
The sqrt of 16 =  4.0
```

NumPy 基础


Application-specific
cesium
PyChrono
MDAnalysis
eht-imaging
iris
khmer
PsychoPy
Qiime2
FiPy
deepchem
nibabel
mne-python
yellowbrick
scikit-HEP
PyWavelets
librosa
SunPy
QuTiP
yt
Domain-specific
Astropy
Astronomy
Biopython
Biology
NLTK
Linguistics
QuantEcon
Economics
cantera
Chemistry
simpeg
Geophysics
Technique-specific
scikit-learn
Machine learning
scikit-image
Image processing
pandas, statsmodels
Statistics
NetworkX
Network analysis
Foundation
SciPy
Algorithms
Matplotlib
Plots
Python
Language
NumPy
Arrays
IPython / Jupyter
Interactive environments
New array implementations
NumPy API
Array Protocols

数据结构：数组 (ndarray)

- bool
- int
- float
- complex
- string



创建数组

numpy.zeros

创建指定大小的数组，
数组元素以 0 来填充：

```
numpy.zeros(shape, dtype = None)
```

```
# 默认为浮点数
x = np.zeros(5)
print(x)
```

```
[0. 0. 0. 0. 0.]
```

```
# 设置类型为整数
y = np.zeros((5,), dtype = np.int)
print(y)
```

```
[0 0 0 0 0]
```

numpy.ones

创建指定形状的数组，
数组元素以 1 来填充：

```
numpy.ones(shape, dtype = None)
```

```
# 默认为浮点数
x = np.ones(5)
print(x)
```

```
[1. 1. 1. 1. 1.]
```

```
# 自定义类型
x = np.ones([2,2], dtype = int)
print(x)
```

```
[[1 1]
 [1 1]]
```

numpy.array

从实际数据中创建数组

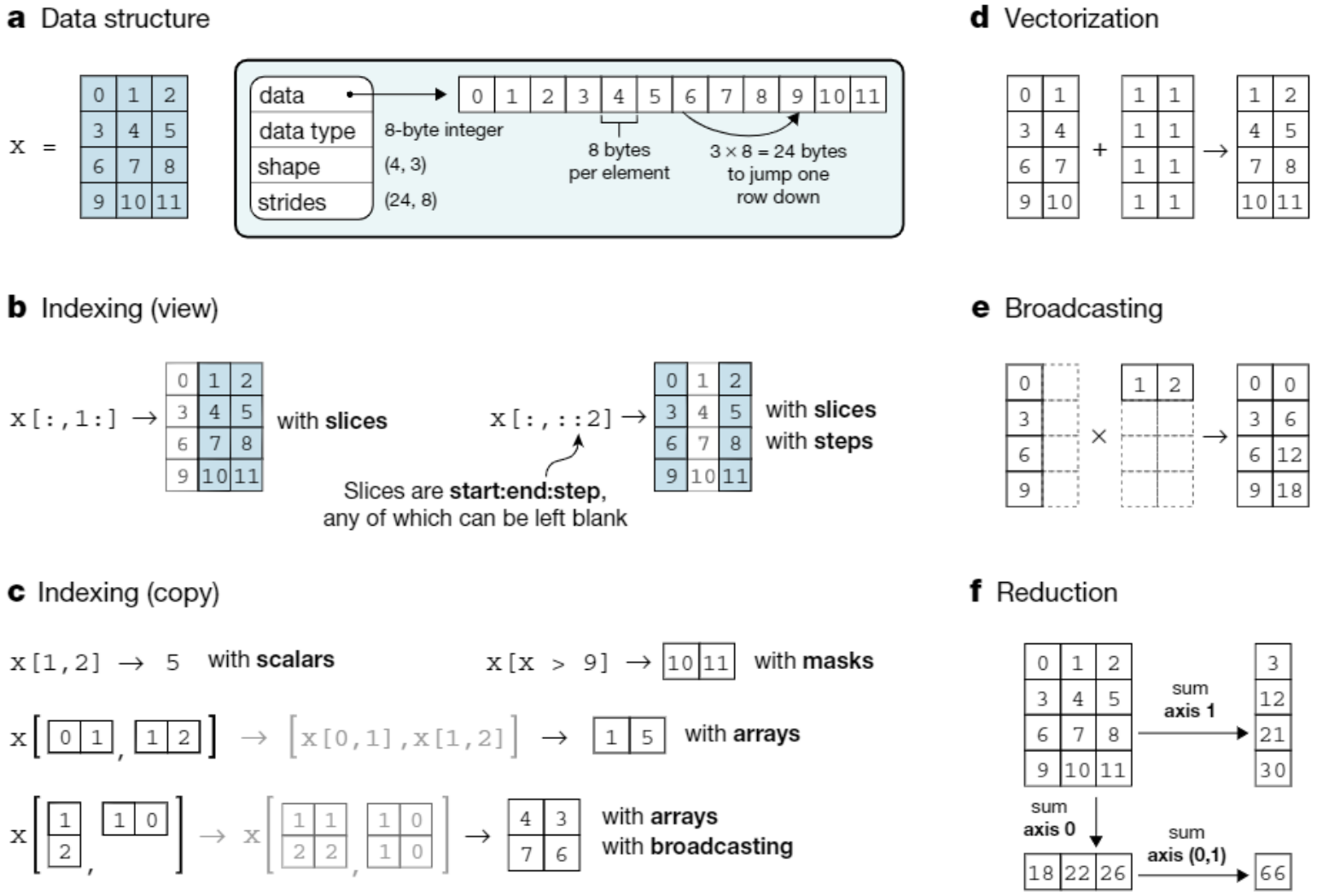
```
In [27]: np.array([[1, 2], [3, 4]])
Out[27]: array([[1, 2],
                [3, 4]])
```

numpy.arange

创建一个序列数组

```
In [30]: np.arange(5)
Out[30]: array([0, 1, 2, 3, 4])
```

数组操作


a Data structure
x =
data
data type
shape
strides
8-byte integer
(4, 3)
(24, 8)
8 bytes per element
3 × 8 = 24 bytes to jump one row down
b Indexing (view)
x[:,1:] →
with slices
x[:,::2] →
with slices
with steps
Slices are start:end:step, any of which can be left blank
c Indexing (copy)
x[1,2] → 5
with scalars
x[x > 9] →
with masks
x[,] → [x[0,1],x[1,2]] →
with arrays
x[,] → x[,] →
with arrays
with broadcasting
d Vectorization
e Broadcasting
f Reduction
sum axis 1
sum axis 0
sum axis (0,1)

数学函数

In [9]:
```
a = np.arange(5)
b = np.array([3, 2, 5, 7, 9])
```

加减乘除

In [10]:
```
a ** 2
```

Out[10]:
```
array([ 0,  1,  4,  9, 16], dtype=int32)
```

In [11]:
```
a + b
```

Out[11]:
```
array([ 3,  3,  7, 10, 13])
```

In [12]:
```
a / b
```

Out[12]:
```
array([0.        , 0.5       , 0.4       , 0.42857143, 0.44444444])
```

In [7]:
```
a = np.array([0,30,45,60,90])
print ('不同角度的正弦值：')
# 通过乘 pi/180 转化为弧度
np.sin(a*np.pi/180)
```

三角函数

```
不同角度的正弦值：
```

Out[7]:
```
array([0.        , 0.5       , 0.70710678, 0.8660254 , 1.        ])
```

In [8]:
```
print ('数组中角度的余弦值：')
np.cos(a*np.pi/180)
```

```
数组中角度的余弦值：
```

Out[8]:
```
array([1.00000000e+00, 8.66025404e-01, 7.07106781e-01, 5.00000000e-01,
       6.12323400e-17])
```

In [16]:
```
a = np.array([[1,2],[3,4]])
b = np.array([[11,12],[13,14]])
# dot 两个数组对应下标元素的乘积和
print(np.dot(a,b))
```

线性代数

```
[[37 40]
 [85 92]]
```

In [17]:
```
a = np.array([[1,2],[3,4]])
b = np.array([[11,12],[13,14]])
# vdot 将数组展开计算内积
# 等价于 1*11 + 2*12 + 3*13 + 4*14 = 130
print (np.vdot(a,b))
```

```
130
```

In [18]:
```
# 一维数组的向量内积
# 等价于 1*0+2*1+3*0
np.inner(np.array([1,2,3]),np.array([0,1,0]))
```

Out[18]:
```
2
```

Math operations

add(x1, x2, /[, out, where, casting, order, ...])
subtract(x1, x2, /[, out, where, casting, ...])
multiply(x1, x2, /[, out, where, casting, ...])
matmul(x1, x2, /[, out, casting, order, ...])
divide(x1, x2, /[, out, where, casting, ...])
logaddexp(x1, x2, /[, out, where, casting, ...])
logaddexp2(x1, x2, /[, out, where, casting, ...])
true_divide(x1, x2, /[, out, where, ...])
floor_divide(x1, x2, /[, out, where, ...])
negative(x, /[, out, where, casting, order, ...])
positive(x, /[, out, where, casting, order, ...])
power(x1, x2, /[, out, where, casting, ...])
float_power(x1, x2, /[, out, where, ...])
remainder(x1, x2, /[, out, where, casting, ...])
mod(x1, x2, /[, out, where, casting, order, ...])
fmod(x1, x2, /[, out, where, casting, ...])
divmod(x1, x2[, out1, out2], / [[, out, ...])
absolute(x, /[, out, where, casting, order, ...])
fabs(x, /[, out, where, casting, order, ...])
rint(x, /[, out, where, casting, order, ...])
sign(x, /[, out, where, casting, order, ...])
heaviside(x1, x2, /[, out, where, casting, ...])
conj(x, /[, out, where, casting, order, ...])
conjugate(x, /[, out, where, casting, ...])
exp(x, /[, out, where, casting, order, ...])
exp2(x, /[, out, where, casting, order, ...])
log(x, /[, out, where, casting, order, ...])
log2(x, /[, out, where, casting, order, ...])
log10(x, /[, out, where, casting, order, ...])
expm1(x, /[, out, where, casting, order, ...])

Trigonometric functions

sin(x, /[, out, where, casting, order, ...])
cos(x, /[, out, where, casting, order, ...])
tan(x, /[, out, where, casting, order, ...])
arcsin(x, /[, out, where, casting, order, ...])
arccos(x, /[, out, where, casting, order, ...])
arctan(x, /[, out, where, casting, order, ...])
arctan2(x1, x2, /[, out, where, casting, ...])
hypot(x1, x2, /[, out, where, casting, ...])
sinh(x, /[, out, where, casting, order, ...])
cosh(x, /[, out, where, casting, order, ...])
tanh(x, /[, out, where, casting, order, ...])
arcsinh(x, /[, out, where, casting, order, ...])
arccosh(x, /[, out, where, casting, order, ...])
arctanh(x, /[, out, where, casting, order, ...])
degrees(x, /[, out, where, casting, order, ...])
radians(x, /[, out, where, casting, order, ...])
deg2rad(x, /[, out, where, casting, order, ...])
rad2deg(x, /[, out, where, casting, order, ...])

Bit-twiddling functions

bitwise_and(x1, x2, /[, out, where, ...])
bitwise_or(x1, x2, /[, out, where, casting, ...])
bitwise_xor(x1, x2, /[, out, where, ...])
invert(x, /[, out, where, casting, order, ...])
left_shift(x1, x2, /[, out, where, casting, ...])
right_shift(x1, x2, /[, out, where, ...])

Comparison functions

logical_and(x1, x2, /[, out, where, ...])
logical_or(x1, x2, /[, out, where, casting, ...])
logical_xor(x1, x2, /[, out, where, ...])
logical_not(x, /[, out, where, casting, ...])
maximum(x1, x2, /[, out, where, casting, ...])
minimum(x1, x2, /[, out, where, casting, ...])
fmax(x1, x2, /[, out, where, casting, ...])
fmin(x1, x2, /[, out, where, casting, ...])

Floating functions

isfinite(x, /[, out, where, casting, order, ...])
isinf(x, /[, out, where, casting, order, ...])
isnan(x, /[, out, where, casting, order, ...])
isnat(x, /[, out, where, casting, order, ...])
fabs(x, /[, out, where, casting, order, ...])
signbit(x, /[, out, where, casting, order, ...])
copysign(x1, x2, /[, out, where, casting, ...])
nextafter(x1, x2, /[, out, where, casting, ...])
spacing(x, /[, out, where, casting, order, ...])
modf(x[, out1, out2], / [[, out, where, ...])
ldexp(x1, x2, /[, out, where, casting, ...])
frexp(x[, out1, out2], / [[, out, where, ...])
fmod(x1, x2, /[, out, where, casting, ...])
floor(x, /[, out, where, casting, order, ...])
ceil(x, /[, out, where, casting, order, ...])
trunc(x, /[, out, where, casting, order, ...])