简介

Python作为一个开源的优秀语言，随着它在数据分析和机器学习方面的优势，已经得到越来越多人的喜爱。据说小学生都要开始学Python了。

Python的优秀之处在于可以安装很多非常强大的lib库，从而进行非常强大的科学计算。

讲真，这么优秀的语言，有没有什么办法可以快速的进行学习呢?

有的，本文就是python3的基础秘籍，看了这本秘籍python3的核心思想就掌握了，文末还有PDF下载链接哦，欢迎大家下载。

Python的主要数据类型

python中所有的值都可以被看做是一个对象Object。每一个对象都有一个类型。

下面是三种最最常用的类型：

- **Integers (int)**

整数类型，比如： **-2, -1, 0, 1, 2, 3, 4, 5**

- **Floating-point numbers (float)**

浮点类型，比如：**-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25**

- **Strings**

字符串类型，比如：“www.flydean.com”

> 注意，字符串是不可变的，如果我们使用**replace()** 或者 **join()** 方法，则会创建新的字符串。

除此之外，还有三种类型，分别是列表，字典和元组。

- **列表**

列表用方括号表示： a_list = [2, 3, 7, None]

- **元组**

元组用圆括号表示： tup=(1,2,3) 或者直接用逗号表示：tup=1,2,3

Python中的String操作

python中String有三种创建方式，分别可以用单引号，双引号和三引号来表示。

基本操作

```
my_string = “Let’s Learn Python!”
another_string = ‘It may seem difficult first, but you can do it!’
a_long_string = ‘’’Yes, you can even master multi-line strings
 that cover more than one line
 with some practice’’’
```

也可以使用print来输出：

```
print(“Let’s print out a string!”)
```

String连接

String可以使用加号进行连接。

```
string_one = “I’m reading “
string_two = “a new great book!”
string_three = string_one + string_two
```

> 注意，加号连接不能连接两种不同的类型，比如String + integer，如果你这样做的话，会报下面的错误：

```
TypeError: Can’t convert ‘int’ object to str implicitly
```

String复制

String可以使用 * 来进行复制操作：

```
‘Alice’ * 5 ‘AliceAliceAliceAliceAlice’
```

或者直接使用print：

```
print(“Alice” * 5)
```

Math操作

我们看下python中的数学操作符：

操作符	含义	举例
**	指数操作	2 ** 3 = 8
%	余数	22 % 8 = 6
//	整数除法	22 // 8 = 2
/	除法	22 / 8 = 2.75
*	乘法	3*3= 9
-	减法	5-2= 3
+	加法	2+2= 4

内置函数

我们前面已经学过了python中内置的函数print（），接下来我们再看其他的几个常用的内置函数：

- **Input() Function**

input用来接收用户输入，所有的输入都是以string形式进行存储：

```
name = input("Hi! What's your name? ")
print("Nice to meet you " + name + "!")
age = input("How old are you ")
print("So, you are already " + str(age) + " years old, " + name + "!")
```

运行结果如下：

```
Hi! What's your name? "Jim"
Nice to meet you, Jim!
How old are you? 25
So, you are already 25 years old, Jim!
```

- **len() Function**

len()用来表示字符串，列表，元组和字典的长度。

举个例子：

```
# testing len()
str1 = "Hope you are enjoying our tutorial!"
print("The length of the string is :", len(str1))
```

输出：

```
The length of the string is: 35
```

- **filter()**

filter从可遍历的对象，比如列表，元组和字典中过滤对应的元素：

```
ages = [5, 12, 17, 18, 24, 32]
def myFunc(x):
  if x < 18:
    return False
  else:
return True
adults = filter(myFunc, ages)
for x in adults:
  print(x)
```

函数Function

python中，函数可以看做是用来执行特定功能的一段代码。

我们使用def来定义函数：

```
def add_numbers(x, y, z):
  a= x + y
  b= x + z
  c= y + z
  print(a, b, c)

add_numbers(1, 2, 3)
```

> 注意，函数的内容要以空格或者tab来进行分隔。

传递参数

函数可以传递参数，并可以通过通过命名参数赋值来传递参数：

```
# Define function with parameters
def product_info(productname, dollars):
    print("productname: " + productname)
    print("Price " + str(dollars))


# Call function with parameters assigned as above
product_info("White T-shirt", 15)

# Call function with keyword arguments
product_info(productname="jeans", dollars=45)
```

列表

列表用来表示有顺序的数据集合。和String不同的是，List是可变的。

看一个list的例子：

```
my_list = [1, 2, 3]
my_list2 = [“a”, “b”, “c”]
my_list3 = [“4”, d, “book”, 5]
```

除此之外，还可以使用**list()** 来对元组进行转换：

```
 alpha_list = list((“1”, “2”, “3”))
 print(alpha_list)
```

添加元素

我们使用**append()** 来添加元素：

```
beta_list = [“apple”, “banana”, “orange”]
beta_list.append(“grape”)
print(beta_list)
```

或者使用**insert()** 来在特定的index添加元素：

```
beta_list = [“apple”, “banana”, “orange”]
beta_list.insert(“2 grape”)
print(beta_list)
```

从list中删除元素

我们使用**remove()** 来删除元素

```
beta_list = [“apple”, “banana”, “orange”]
beta_list.remove(“apple”)
print(beta_list)
```

或者使用**pop()** 来删除最后的元素：

```
beta_list = [“apple”, “banana”, “orange”]
beta_list.pop()
print(beta_list)
```

或者使用**del** 来删除具体的元素：

```
beta_list = [“apple”, “banana”, “orange”]
del beta_list [1]
print(beta_list)
```

合并list

我们可以使用+来合并两个list：

```
my_list = [1, 2, 3]
my_list2 = [“a”, “b”, “c”]
combo_list = my_list + my_list2
combo_list
[1, 2, 3, ‘a’, ‘b’, ‘c’]
```

创建嵌套的list

我们还可以在list中创建list：

```
my_nested_list = [my_list, my_list2]
my_nested_list
[[1, 2, 3], [‘a’, ‘b’, ‘c’]]
```

list排序

我们使用sort()来进行list排序：

```
alpha_list = [34, 23, 67, 100, 88, 2]
alpha_list.sort()
alpha_list
[2, 23, 34, 67, 88, 100]
```

list切片

我们使用[x:y]来进行list切片：

```
 alpha_list[0:4]
[2, 23, 34, 67]
```

修改list的值

我们可以通过index来改变list的值：

```
beta_list = [“apple”, “banana”, “orange”]
beta_list[1] = “pear”
print(beta_list)
```

输出：

```
[‘apple’, ‘pear’, ‘cherry’]
```

list遍历

我们使用**for loop** 来进行list的遍历：

```
for x in range(1,4):
  beta_list += [‘fruit’]
  print(beta_list)
```

list拷贝

可以使用**copy()** 来进行list的拷贝：

```
beta_list = [“apple”, “banana”, “orange”]
beta_list = beta_list.copy()
print(beta_list)
```

或者使用list()来拷贝：

```
beta_list = [“apple”, “banana”, “orange”]
beta_list = list (beta_list)
print(beta_list)
```

list高级操作

list还可以进行一些高级操作：

```
list_variable = [x for x in iterable]

 number_list = [x ** 2 for x in range(10) if x % 2 == 0]
 print(number_list)
```

元组

元组的英文名叫Tuples，和list不同的是，元组是不能被修改的。并且元组的速度会比list要快。

看下怎么创建元组：

```
my_tuple = (1, 2, 3, 4, 5)
my_tuple[0:3]
(1, 2, 3)
```

元组切片

```
 numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
 print(numbers[1:11:2])
```

输出：

```
(1,3,5,7,9)
```

元组转为list

可以使用list和tuple进行相互转换：

```
x = (“apple”, “orange”, “pear”)
y = list(x)
y[1] = “grape”
x = tuple(y)
print(x)
```

字典

字典是一个key-value的集合。

在python中key可以是String，Boolean或者integer类型：

```
 Customer 1= {‘username’: ‘john-sea’, ‘online’: false, ‘friends’:100}
```

创建字典

下面是两种创建空字典的方式：

```
new_dict = {}
other_dict= dict()
```

或者像下面这样来初始赋值：

```
new_dict = {
  “brand”: “Honda”,
  “model”: “Civic”,
  “year”: 1995
}
print(new_dict)
```

访问字典的元素

我们这样访问字典：

```
x = new_dict[“brand”]
```

或者使用`dict.keys()`，`dict.values()`，`dict.items()`来获取要访问的元素。

修改字典的元素

我们可以这样修改字典的元素：

```
#Change the “year” to 2020:
new_dict= {
  “brand”: “Honda”,
  “model”: “Civic”,
  “year”: 1995
}
new_dict[“year”] = 2020
```

遍历字典

看下怎么遍历字典：

```
#print all key names in the dictionary
for x in new_dict:
  print(x)
#print all values in the dictionary
for x in new_dict:
  print(new_dict[x])
#loop through both keys and values
for x, y in my_dict.items(): print(x, y)
```

if语句

和其他的语言一样，python也支持基本的逻辑判断语句：

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

看下在python中if语句是怎么使用的：

```
if 5 > 1:
  print(“That’s True!”)
```

if语句还可以嵌套：

```
x = 35
  if x > 20:
    print(“Above twenty,”)
    if x > 30:
      print(“and also above 30!”)
```

elif:

```
a = 45
b = 45
if b > a:
  print(“b is greater than a”)
elif a == b:
  print(“a and b are equal”)
```

if else:

```
if age < 4:
ticket_price = 0
elif age < 18:
ticket_price = 10
else: ticket_price = 15
```

if not:

```
new_list = [1, 2, 3, 4]
x = 10
if x not in new_list:
  print(“’x’ isn’t on the list, so this is True!”)
```

Pass:

```
a = 33
b = 200
if b > a:
  pass
```

Python循环

python支持两种类型的循环，for和while

for循环

```
 for x in “apple”:
    print(x)
```

for可以遍历list, tuple,dictionary,string等等。

while循环

```
#print as long as x is less than 8
i =1
while i< 8:
  print(x)
  i += 1
```

break loop

```
i =1
while i < 8:
  print(i)
  if i == 4:
    break
  i += 1
```

Class

Python作为一个面向对象的编程语言，几乎所有的元素都可以看做是一个对象。对象可以看做是Class的实例。

接下来我们来看一下class的基本操作。

创建class

```
 class TestClass:
    z =5
```

上面的例子我们定义了一个class，并且指定了它的一个属性z。

创建Object

```
 p1 = TestClass()
 print(p1.x)
```

还可以给class分配不同的属性和方法：

```
class car(object):
  """docstring"""
  def __init__(self, color, doors, tires):
    """Constructor"""
    self.color = color
    self.doors = doors
    self.tires = tires

  def brake(self):
```

```
    """
    Stop the car
    """
    return "Braking"

  def drive(self):
    """
    Drive the car
    """
    return "I'm driving!"
```

创建子类

每一个class都可以子类化

```
class Car(Vehicle):
  """
  The Car class
  """

def brake(self):
  """
  Override brake method
  """
  return "The car class is breaking slowly!"

if __name__ == "__main__":
  car = Car("yellow", 2, 4, "car")
  car.brake()
  'The car class is breaking slowly!'
  car.drive()
  "I'm driving a yellow car!"
```

异常

python有内置的异常处理机制，用来处理程序中的异常信息。

内置异常类型

- AttributeError — 属性引用和赋值异常
- IOError — IO异常
- ImportError — import异常
- IndexError — index超出范围
- KeyError — 字典中的key不存在
- KeyboardInterrupt — Control-C 或者 Delete时，报的异常
- NameError — 找不到 local 或者 global 的name
- OSError — 系统相关的错误

- SyntaxError — 解释器异常
- TypeError — 类型操作错误
- ValueError — 内置操作参数类型正确，但是value不对。
- ZeroDivisionError — 除0错误

异常处理

使用try，catch来处理异常：

```
my_dict = {“a”:1, “b”:2, “c”:3}
try:
    value = my_dict[“d”]
except KeyError:
    print(“That key does not exist!”)
```

处理多个异常：

```
my_dict = {“a”:1, “b”:2, “c”:3}
try:
  value = my_dict[“d”]
except IndexError:
  print(“This index does not exist!”)
except KeyError:
  print(“This key is not in the dictionary!”)
except:
  print(“Some other problem happened!”)
```

try/except else:

```
my_dict = {“a”:1, “b”:2, “c”:3}
try:
  value = my_dict[“a”]
except KeyError:
  print(“A KeyError occurred!”)
else:
  print(“No error occurred!”)
```