


# python 基础

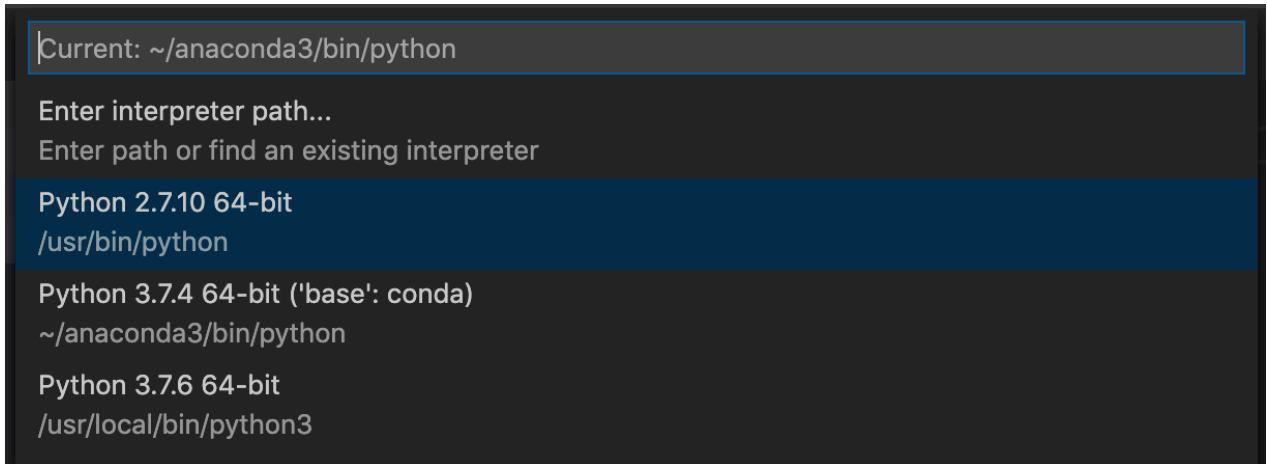
---

python: 解释型语言

VS Code 安装python扩展

选择python解释器:  打开命令面板 -> python: select interpreter 搜索并选择

avatar



新建文件夹, 新建文件并命名为.py: 告诉VS Code将这个文件解释为Python程序

## 变量和简单数据类型

---

### 字符串

```
string = "whatever you want to print"
```

字符串拼接: `str3 = str1 + str2`

截取字符串: `new_str = str[a:b]`, `[a:b]` 左闭右开

字符串添加tab: `\t`

字符串添加换行: `\n`

删除字符串结尾的多余空白: `new_str = str.rstrip()`, `str` 中仍存有多余空白

删除字符串开头的多余空白: `new_str = str.lstrip()`

删除字符串中多余空白: `new_str = str.strip()`

输出字符串: `print(str)` Or `print("...")`

### 数字

```
n = 23
```

`**`: 乘方运算

## 列表

---

Python有6个序列的内置类型，但最常见的是列表和元组。

序列都可以进行的操作包括索引，切片，加，乘，检查成员。

创建列表：`list = []` or `list = [obj1,obj2,obj3,obj4]`

访问左数第n个元素：`list[n]`，最左边的元素：`list[0]`

访问右数第n个元素：`list[-n]`，最右边的元素：`list[-1]`

向列表中添加元素：`list.append(obj)`

向列表中插入元素：`list.insert(i,obj)`

从列表中删除元素：`del list[i]`

从列表中删除并返回该元素的值：`list.pop(i)` 默认：`i=-1`

根据值删除元素：`list.remove(obj)` `remove()` 只删除第一个指定的值。如果要删除的值可能在列表中出现多次，就需要使用循环来判断是否删除了所有这样的值。

列表永久性排序：`list.sort(key=None, reverse=False)` 默认升序，若逆序：`reverse=True`

临时排序：`sorted(list)`，此时list仍为未排序的顺序

反转列表：`list.reverse()`

求列表长度：`len(list)`

## 操作列表

---

循环遍历

```
for obj in list:
    print(obj)
print("iterate finish.")
```

创建数值列表：`range(a:b)`左闭右开

```
l = []
for i in range(1:5):
    l.append(i)
```

或：

```
numbers = list(range(1,5))
squares = [values**2 for values in range (1,5)]
```

若指定步长为2：

```
numbers = list(range(1,5,2))
```

## 切片

```
new_list = list[from:to]
```

若未指定from, 即 `list[:to]`: 默认从头开始

若要切片终止于结尾元素: `new_list = list[from:]`

若要最后i个元素: `list[-i:]`

复制全部列表: `l2 = l1[:]`

复制部分列表: `l2 = l1[from:to]`

若直接 `l2 = l1`: 两个变量指向同一个列表

## 元组

元组与列表类似, 不同之处在于元组的元素不能修改; 元组使用小括号, 列表使用方括号。

```
tuple = (obj1,obj2,...)
```

元组中只包含一个元素时, 需要在元素后面添加逗号, 否则括号会被当作运算符使用: `tup = (obj1,)`

元组元素不能修改, 但元组本身可重新赋值, 也可拼接。

## if

检查多个条件: `and` 或者 `or`

检查特定值是否包含在列表中: `"value" in list` 或者 `"value" not in list`

结构:

```
if condition1:
    statement1
elif condition2:
    statement2
else condition3:
    statement3
```

## 字典

字典是一系列键-值对。每个键都与一个值相关联, 可以使用键来访问与之相关联的值。与键相关联的值可以是数字、字符串、列表乃至字典。事实上, 可将任何Python对象用作字典中的值。

字典的每个键值(key=>value)对用冒号(:)分割, 每个对之间用逗号(,)分割, 整个字典包括在花括号({})中:

```
dict = {"color":"green", "size":"large", "score":5}
```

创建空字典: `dict = {}`

添加键-值对: `dict['colors'] = "green"`

修改字典中的值: `dict['colors'] = "new_color"`

删除键-值对: `del dict['key']`

遍历字典:

```
for key, values in dict.items():
    print("key:"+key + " " + "value:"+value)
```

遍历字典中的键: `for name in dict.keys():`

按顺序遍历字典中的键: `for name in sorted(dict.keys()):`

遍历字典中的值: `for val in dict.values():`

## 嵌套

字典列表:

```
dict1 = {...}
dict2 = {...}
dict3 = {...}
dic_list = [dict1, dict2, dict3]
```

在字典中存储列表:

```
dict = {"key":[str1,str2], ...}
```

在字典中存储字典:

```
users = { 'einstein': {
    'first': 'albert', 'last': 'einstein', 'location': 'princeton', },
    'curie': {
    'first': 'marie', 'last': 'curie', 'location': 'paris', },
}
```

## 用户输入

函数input() 接受一个参数: 即要向用户显示的提示或说明, 让用户知道该如何做:

```
message = input("Tell me something, and I will repeat it back to you: ")
```

使用函数input() 时, Python将用户输入解读为字符串, 使用int() 来获取数值输入

```
age = input("How old are you? ")
age = int(age)
```

## 函数

```
def function(parameters):
    statements
```

关键字实参：是传递给函数的名称-值对。

直接在实参中将名称和值关联起来了，无需考虑函数调用中的实参顺序，还清楚地指出了函数调用中各个值的用途。使用关键字实参时，务必准确地指定函数定义中的形参名。

编写函数时，可给每个形参指定默认值 e.g. `def sort(reverse=False)`

可选实参： `def func(par='')` 设置默认为空

参数可为任意对象：列表、字典..... 在 python 中，类型属于对象，变量无类型

返回值可为任意对象：列表、字典.....

## 可更改对象与不可更改对象

在 python 中，strings、tuples和 numbers 是不可更改的对象，而 list,dict 等则是可以修改的对象。

不可变类型immutable：

变量赋值 a=5 后再赋值 a=10，这里实际是新生成一个 int 值对象 10，再让 a 指向它，而 5 被丢弃，不是改变a的值，相当于新生成了a。

类似 c++ 的值传递

可变类型mutable：

变量赋值 la=[1,2,3,4] 后再赋值 la[2]=5 则是将 list la 的第三个元素值更改，本身la没有动，只是其内部的一部分值被修改了。

类似 c++ 的引用传递

若要禁止函数修改列表，要将列表的副本传递给函数：`def func(list[:])`

## 传递任意数量的实参

`def func(*tuples)：`

形参名\*tuples 中的星号让Python创建一个名为toppings 的空元组，并将收到的所有值都封装到这个元组中。

`def func(**dicts)：`

形参\*\*dicts 中的两个星号让Python创建一个名为dicts的空字典，并将收到的所有名称-值对都封装到这个字典中。在这个函数中，可以像访问其他字典那样访问dicts中的名称-值对。

## 将函数存储到模块

模块是扩展名为.py的文件，包含要导入到程序中的代码。

将函数存储在被称为模块 的独立文件中，再将模块导入 到主程序中。import 语句允许在当前运行的程序文件中使用模块中的代码。

导入模块：`import module`

导入模块中所有函数：`from module import *`

导入特定函数：`from module import function`

指定函数别名：`from module import function as f`

指定模块别名：`import numpy as np`

## 类

---

```
class NewClass()  
    def __init__(self, p1, p2):  
        self.v1 = p1  
        self.v2 = p2  
        self.v3 = 0 #默认v3属性的值为0
```

方法**init()**：类似C++中构造函数，必须包含形参self。

Python调用这个**init()** 方法来创建NewClass实例时，将自动传入实参self 。每个与类相关联的方法调用都自动传递实参self，它是一个指向实例本身的引用，让实例能够访问类中的属性和方法。

根据类创建实例：

```
obj1 = NewClass(p1,p2)
```

方法**init()** 并未显式地包含return 语句，但Python自动返回一个表示这个类的实例。

## 继承

```
class Car()  
    def __init__(self,p1,p2,p3):  
        self.v1 = p1  
        self.v2 = p2  
        self.v3 = p3
```

ElectricCar继承Car：

```
class ElectricCar(car)  
    def __init__(self,p1,p2,p3):  
        super().__init__(p1,p2,p3)  
        self.battery_size = 70
```

将实例用作属性：

```
class Battery()  
    def __init__(self,battery_size=70):  
        self.battery_size=battery_size
```

在ElectricCar中实例化Battery：

```
class ElectricCar(Car)  
    def __init__(self,p1,p2,p3):  
        super().__init__(p1,p2,p3)  
        self.battery = Battery()
```

## 文件和异常

---

## 读取文件

```
with open(file_dir) as file_obj:
    contents = file_obj.read()
```

创建包含文件各行内容的列表：

```
with open(file_dir) as f:
    lines = f.readlines()
for line in lines:
    print(line.rstrip())
```

## 写入文件

写入空文件：

```
with open(emptyFile, "w") as f:
    f.write(msg)
```

添加到已有文件末尾而不是覆盖原有内容：

```
with open(file, "a") as f:
    f.write(msg)
```