# python基础学习笔记

## 引入

```
print("Hello World")
```

## 字符串

```
# 字符串打印
sentence = "Hello World"
print(sentence)

# 字符串输入并打印
Name = input("Please Enter Your Name:")
print(Name)

# 字符串的比较 （相等比较为“==”不等比较为“!=”）
if sentence_1 == sentence_2:
    print(True)
else:
    print(False)

# 字符串中的字符
Cut_Word = "Hello Python"
Cut0 = Cut_Word[0]  # 字符串数组下标从0开始
print(Cut0)     # 输出'H'
Cut1 = Cut_Word[8]
print(Cut1)     # 输出‘t’

# 字符串切片
Cut_Word = "Hello Python"
Cut0 = Cut_Word[0:5]    # 从0开始，到最后一个下标前面一个字符结束
print(Cut0)
```

## 变量

**Python变量类型包括：**

- 整数
- 浮点
- 布尔值（真或假）

同时，变量之间可以进行加减乘除等数学运算

```python
# 变量的定义
x = 1
y = 2.501
z = True

# 用户输入并进行数据类型转换（常用的转换函数有：float() 和 int() ）
x = int(input("Please Enter the First Number:"))
y = int(input("Please Enter the Second Number:"))
sum_ = x + y
print(sum_)
```

## 列表

- python中列表相当于其他语言中的数组，但是具有其它功能
- Python中的列表使用 `[ ]` 代表

```python
# Python 列表的使用
L = [1, "Hello", 3, "Append"]
print(L[1])     # 输出 Helloda

# 使用 append 和 remove 函数来操作列表
print(L)
L.append('Python')
L.append('Coding')
print(L)
L.remove(1)
print(L)

# 使用 sort 和 reverse 函数对列表进行排序，其中sort函数为正序，reverse为逆序
L = [4, 2, 6, 1, 0, 2]
L.sort()    # sort正向排序
print(L)    # 输出：[0, 1, 2, 2, 4, 6]
L.reverse()     # reverse逆向排序
print(L)    # 输出：[6, 4, 2, 2, 1, 0]
```

## if语句

```python
# if语句的使用 可以使用 and 和 or 函数来嵌套多个判断
age = 21

# demo 1
sentence = "Guess my age, you only have one chance!"
print(sentence)
age_input = int(input("Input:"))
if age_input == age :
    Flag = True
else:
```

```
    Flag = False
print(Flag)

# demo 2
print(sentence)
age_input = int(input("Input:"))
if (age_input > age-10) and (age_input < age + 10):
    Flag = True
else:
    Flag = False
print(Flag)
```

## 函数

```
# 函数的定义
def function(parameters):
    instructions
    return value

# demo
def f():
    x = int(input("Please input the first num:"))
    y = int(input("Please input the second num:"))
    product = x * y
    return product


res = f()
print(res)
```

## 局部和全局变量

```
# 将局部变量转换成全局变量 global 函数
global x

# demo
z = 10

def func1():
    global z
    z = 3

def func2(x, y):
    global z
    return x+y+z

func1()
total = func2(4, 5)
```

```
print(total)
```

## for、while循环

```
# for 循环
items = ["Python", "I", "Love", "Coding"]

for item in items:
    print(item)

# while 循环
Guess_Num = 21
Current_Num = 0

while Guess_Num != Current_Num:
    Current_Num = int(input("Please input a Num:"))

    if Guess_Num != Current_Num:
        print("False")

print("Correct!")
```

## Python范围　（range函数）

```
# range 函数使用示例
range(lower_bound, upper_bound, step_size)
```

## 元组

```
# 元组的定义
tuple1 = () # 空元组
tuple2 = (3,)   # 单个元素的元组必须在末尾处加上逗号
tuple3 = (3,1,8)    # 多个元素时不需要在末尾处添加都好哦

# 元组中的元素不可修改，只可以在末尾添加 使用 +
x = (1,2,3)
y = x + (2,3)
print(y)

# 元组的索引和存储
print(y[1])
name1, name2, name3, name4 = ("Cris", "Hi", "Kris", "Peng")
print(name3)
```

# 字典

使用大括号 `{ }` 创建一个字典，每个元素都可以映射到某个值，整数或者字符串可用于索引

```python
# 字典的创建
x = {'a': 1, 'b': 2, 'c': 3, 'b': 4}     # 字典不允许两个相同的键，若两个相同，则后面的被记录
print(x['b'])    # 输出4

# 字典的添加和删除
x["Python"] = "WuDi"     # 字典的添加
print(x)

del x["Python"]      # 字典的删除
print(x)
```

# 数据类型转换

| 函数 | 描述 |
| --- | --- |
| `int(x)` | 将 `x` 转换为整数 |
| `long(x)` | 将 `x` 转换为长整数 |
| `float(x)` | 将 `x` 转换为浮点数 |
| `str(x)` | 将 `x` 转换为字符串。 `x` 可以是 `float` 类型。 整数或长整数。 |
| `hex(x)` | 将 `x` 整数转换为十六进制字符串 |
| `asc(x)` | 将 `x` 整数转换为字符 |
| `ord(x)` | 将字符 `x` 转换为整数 |

```python
# 数据类型的转换
x = 12
y = 12.34567
print("X = " + str(x))
print("Y = " + str(y))
```

# 随机数字的生成

```python
# 随机数的生成 使用 random 包中的函数

# random 函数生成 0-1 之间的随机数
from random import *
print(random()) # 输出结果为0-1的随机数

# randint 函数生成范围内的整数随机数
x = randint(1,100)  # 产生1-100之间的随机数（整数）

# uniform 函数生成范围内的浮点数随机数
y = uniform(1,50)   # 产生1-50之间的随机数（浮点数）
```

# 读取文件

- 文件读取的函数： `read()` `readline()` `readlines`
- 文件读取的步骤： 首先 `open()` 函数打开文件，然后读取文件，最后 `close()` ，关闭文件

```python
# 打开文件
f = open("D:\系统默认\桌面\python text.txt", 'r', encoding="utf-8")
print(f.name)   # 打印文件的名字

# 文件的读取方法 read函数
content = f.read()
print(content)

# readline函数读取文件
content = f.readline()
while content:
    content = f.readline()
    print(content)

# readlines函数读取文件
content = f.readlines()
print(content)  # 输出结果为一个列表

# 关闭文件
f.close()
```

with语句能自动处理上下文环境产生的异常并且关闭文件句柄

```python
with open('D:\系统默认\桌面\python text.txt', encoding='utf-8') as f2:
    #读取文件
    contents = f2.read()
print(contents)
```

# 写入文件

```
# 需要写入的文件 （会覆盖原文件）
filename = "D:\系统默认\桌面\myfile.txt"

# 以写入的方式打开文件
myfile = open(filename, 'w')    # 如果想要附加在原文件的后面，则用 myfile = open(filename, 'a')

# 写入数据
myfile.write('Written with Python\n')

# 关闭文件
myfile.close()
```

# 对象和类

区分语句、函数、类，三者之间的区别：

- 语句：在编写程序的过程中，不同的语句代表不同的命令
- 函数：可重用的语句组，有助于结构化代码并提高可读性
- 类：用于创建具有功能和变量的对象，这种样式通常称为 `面向对象编程` ，例如，字符串是对象的实例，具有如book.replace() `和` book.lowercase()等功能

python中创建虚拟对象。虚拟对象中包括变量和方法。根据类来创建对象称为**实例化。**根据约定，在python中，**首字母大写的名称指的是类。**
**每种方法中必须包含 `__init__` 方法，每个方法中必须包含self变量。同时，每次调用实例时，都会执行 `__init` 方法**
实例代码：

```
class Dog:
    """一次模拟小狗的尝试"""

    def __init__(self, name, age):
        """初始化属性 name和 age"""
        self.name = name
        self.age = age

    def sit(self):
        """模拟小狗收到命令时蹲下"""
        print(self.name + " is now sitting!")

    def roll_over(self):
        """模拟小狗收到命令时打滚"""
        print(self.name + " is now rolling over!")


my_dog = Dog("Kimi", 9)      # 根据类创建一个实例
his_dog = Dog("Kris", 5)     # 根据类创建第二个实例
print("My dog's name is " + str(my_dog.name))
print("My dog's age is " + str(my_dog.age))
```

```
my_dog.sit()
my_dog.roll_over()
```

```
# 类的尝试
class CoffeeMachine:
    """模拟咖啡机的尝试"""
    beans = 0
    water = 0

    def __init__(self, beans, water):
        self.beans = beans
        self.water = water

    def AddBean(self):
        self.beans = self.beans + 1

    def RemoveBean(self):
        self.beans = self.beans - 1

    def AddWater(self):
        self.water = self.water + 1

    def RemoveWater(self):
        self.water = self.water - 1


PythonBeans = CoffeeMachine(30, 30)
for i in range(5):
    PythonBeans.AddBean()
for i in range(10):
    PythonBeans.RemoveWater()
for i in range(8):
    PythonBeans.RemoveBean()
for i in range(7):
    PythonBeans.AddWater()

# 结果展示
print(PythonBeans.water)    # 输出结果为27
print(PythonBeans.beans)    # 输出结果为27
```

## 封装

概念：限制对方法和变量的访问，防止意外修改数据
注意： 封装可以防止意外访问，但是不能防止有意访问

| 类型 | 描述 |
| --- | --- |
| 公共方法 | 可从任何地方访问 |
| 私有方法 | 仅在自己的课程中可访问。 以两个下划线开头 |
| 公共变量 | 可从任何地方访问 |
| 私有变量 | 仅在自己的类或方法（如果已定义）中可访问。 以两个下划线开头 |

```python
# 封装
class Car:
    __origin_oil = 0
    def __init__(self, oil):
        self.__updateSoftware()
        self.oil = oil

    def drive(self):
        print("Driving!")

    def addoil(self):
        self.oil += 1

    def __updateSoftware(self): # 私有方法
        print("Updating software!")


redcar = Car()
redcar.drive()
```

## 方法重载

概念：可以用多种方式来调用它定义的方法，可以使用0个、1个或者多个参数来调用它。这种叫做方法重载，不是所有语言都支持，但是python支持

```python
# 方法重载示例

class Human:
    def __init__(self, name=None):  # name的初始值为空，调用的时候就可不用直接赋值
        self.name = name

    def sayhello(self):
        print("Hello World! " + str(self.name))

    def test(self):
        pass    # 再编程的过程中，如果该方法暂时不需要补充，可以使用pass语句暂时跳过


Kris = Human()
Kris.sayhello()
```

```
Kris.name = "Ailice"
Kris.sayhello()
```

# 继承

类可以集成其他类的功能。如果使用从超类集成的类创建对象，则该对象将包含该类和超类的方法和变量。与其他语言不同的是，python支持从多个类继承。

`super()` 函数：一个特殊的函数，该函数使之能够调用父类的方法。示例中的代码表示，让python调用Car类的方法 `__init__` ，让ElectricCar实例包含这个方法中定义的所有属性。父类又称为超类(supercalss)，名称super就是由此而来

```
# 继承

# 父类
class Car:
    """一次模拟汽车的尝试"""

    def __init__(self, make, model, year):
        """初始化描述汽车的属性"""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """返回整洁的描述性信息"""
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()    # title()、upper()、lower()三个函数


my_new_car = Car("audi", "a4", 2020)
print(my_new_car.get_descriptive_name())


# 子类
class ElectricCar(Car): # ElectricCar为子类，Car为父类
    """电动汽车的独特之处"""

    def __init__(self, make, model, year):
        """初始化父类的属性"""
        super().__init__(make, model, year)
        self.battery_size = 75


my_tesla = ElectricCar("tesla", "model3", 2023)
print(my_tesla.get_descriptive_name())
```

类的导入

```python
from car import Car # 第一个car为文件名，第二个Car为类名
from car import Car, ElectricCar    # 从一个模块中导入多个类，使用逗号分隔
import car  # 导入整个模块文件
from car import *   # 导入整个模块文件

# 使用
my_car = car.Car("tesla", "roadster", 2019) # 第一个为文件名，第二个为类名

# 别名的使用
from car import ElectricCar as EC
import pandas as pd
```

## 多态

```python
# 多态
class Animal:
    def make_sound(self):
        pass


class Dog:
    def make_sound(self):
        print("汪汪汪！")


class Cat:
    def make_sound(self):
        print("喵喵喵！")


class Ox:
    def make_sound(self):
        print("牛叫！")


def animal_sound(animal):   # 使用一个函数实现多个类中共同的属性
    animal.make_sound()


my_dog = Dog()
animal_sound(my_dog)
```

# python的主函数

所有语句都从main函数开始执行，但是一个良好的程序应该从main函数开始执行

```
if __name__ == "__main__" :
        Logic Statements
```

# 递归

```
# 阶乘的递归实现
def fac(num):
    if num == 0:
        return 1
    elif num == 1:
        return 1
    else:
        return num * fac(num-1)


if __name__ == "__main__":
    res = fac(3)
    print(res)  # 输出结果为6


# 当递归次数过多的时候，程序可能会崩溃，此时可以使用以下命令来增大最大的递归次数
import sys
sys.setrecursionlimit(num)  # num 为允许的最大递归次数
```

# 魔术方法

定义：使用类时，此方法会自动执行。例如类中的 `__init__()` 方法、 `__str__()` 方法、 `__del__()` 方法

- `__init__` 方法：初始化操作，实例化对象是，其中的参数会自动执行，相当于初始化操作
- `__str__` 方法：当时用print打印对象时，会默认打印对象的内存地址，如果定义了该方法，则会return其中定义的数据
- `__del__` 方法：与 `__init__` 方法是一对，在调用 `del` 删除对象后，会执行该方法

```
class Person(object):   # object会继承的父类
    def __init__(self):
        print("此为魔法方法，会自动执行")

    Cris = Person()

# 定义一个类
class Car():
    # 首先定义一个__init__方法，用于初始化实例对象属性
```

```
    def __init__(self, brand, model, color):
        self.brand = brand
        self.model = model
        self.color = color

    # 定义一个__str__内置魔术方法，用于输出小汽车的相关信息
    def __str__(self):
        return f'汽车品牌：{self.brand}，汽车型号：{self.model}，汽车颜色：
{self.color}'

# 实例化对象c1
c1 = Car('奔驰', 'S600', '黑色')
print(c1)

# 1、定义一个类
class Car():
    # 首先定义一个__init__方法，用于初始化实例对象属性
    def __init__(self, brand, model, color):
        self.brand = brand
        self.model = model
        self.color = color

    # 定义一个__str__内置魔术方法，用于输出小汽车的相关信息
    def __str__(self):
        return f'汽车品牌：{self.brand}，汽车型号：{self.model}，汽车颜色：
{self.color}'

# 2、实例化对象c1
c1 = Car('奔驰', 'S600', '黑色')
print(c1)
```

## 函数方法接收多个参数

Python的自定义函数中，如果需要传入的实际参数有不定数量个，可以使用 `*parameter` ，和 `**parameter`

```
# *parameter: 接收任意多个实际参数并将其放在一个元组中
def printcoff(*para):
    for item in para:
        print(item)
printcoff("karl","inter","killer")
>>>
karl
inter
killer

# **parameter: 接收任意多个参数关键字和参数，并将其放到一个字典中
def printcoff(**para):
    for key, value  in para.items():
        print(key,value)
pdict = {"1":"karl","2":"inter","3":"killer","4":"python"}
printcoff(**pdict)
>>>
1 karl
```

```
2 inter
3 killer
4 python
```