Python核心语法速查

1. 数据基本运算

人机交互

软件开发基本步骤

```
数据 = input("提示信息")
print("结果")
```

数据

变量

操作数据

```
变量名称 = 数据
变量名称1=变量名称2=数据
变量名称1，变量名称2 = 数据1，数据2
del 变量名
```

数据类型

```
str 类型：'悟空'
int 类型：123
float 类型：1.2
bool 类型：True  False
结果 = 目标类型(待转数据)
```

运算

算数运算符

```
加法：数值 + 数值
减法：数值 + 数值
乘法：数值 * 数值
幂运算：数值 ** 数值
除法
    小数商：数值 / 数值
    整数商：数值 // 数值
    取余数：数值 % 数值
```

增强运算符

```
累加：变量 += 数值
累减：变量 -= 数值
累乘：变量 *= 数值
累幂运算：变量 **= 数值
累除
    小数商：变量 /= 数值
    整数商：变量 //= 数值
    取余数：变量 %= 数值
```

比较运算符

```
大于：数值 > 数值
小于：数值 < 数值
等于：数据 == 数据
不等于：数据 != 数据
大于等于：数值 >= 数值
小于等于：数值 <= 数值
```

逻辑运算符

```
条件 and 条件
两个条件同时满足
条件 or 条件
两个条件满足其一即可
not 条件
```

身份运算符

判断地址

```
变量 is 变量
变量 is not 变量
```

2. 流程控制语句

选择语句

```
if 条件:
    满足条件执行
else:
    不满足条件执行

if 条件1:
    满足条件1执行
elif 条件2:
    不满足条件1但满足条件2执行
else:
    不满足以上所有条件执行
```

循环语句

```
while 条件:
    循环体
```
根据条件重复

```
for 变量 in 可迭代对象:
    变量存储可迭代对象中的元素
```
取出全部元素

```
for 变量 in range(开始,结束,间隔):
    变量存储整数
```
根据次数重复

跳转语句

```
break
```
跳出循环

```
continue
```
跳过循环

3. 常用容器类型

字符串

存储字符编码的不可变序列容器

```
"%s" % 数据
"%.2d" % 整数
"%.2f" % 小数
f"{数据:格式}"
```

列表

存储变量的可变序列容器

基本操作

1 创建

```
列表名 = [数据1,数据2]
列表名 = list(可迭代对象)
```

2 添加

```
列表名.append(元素)
列表名.insert(索引,元素)
```

3 定位

```
列表名[整数]
列表名[开始：结束：间隔]
```

4 删除

```
del 列表名[索引或切片]
列表名.remove(元素)
```

5 遍历

```
for item in 列表名:
for i in range(len(列表名)):
```

元组

存储变量的不可变序列容器

基本操作

1 创建

```
元组名 = (数据1,数据2)
元组名 = tuple(可迭代对象)
```

2 定位

```
元组名[整数]
元组名[开始：结束：间隔]
```

3 遍历

```
for item in 元组名:
for i in range(len(元组名)):
```

4 注意

```
元组名 = 元素1,元素2,元素3
元组名 = (元素,)
变量1，变量2 = 序列名
```

字典

存储键值对的可变散列容器

基本操作

1 创建

```
字典名 = { 键1:值，键2:值 }
字典名 = dict(可迭代对象)
```
可迭代对象中元素必须能够一分为二

2 添加

```
字典名[键] = 值
```

3 定位

```
字典名[键]
```

4 删除

```
del 字典名[键]
```

5 遍历

```
for key in 字典名:
for value in 字典名.values():
for key,value in 字典名.items():
```

4. 函数

定义

```
def 函数名(参数):
    函数体
    return 返回值
```

调用

```
变量 = 函数名(数据)
```

参数

形参

```
def 函数名(参数名1,参数名2)
```
位置形参 实参必填

```
def 函数名(参数名1=数据,参数名2=数据)
```
默认形参 实参可选

```
def 函数名(*args)
```
星号元组形参 位置实参数量无限

```
def 函数名(**kwargs)
```
双星号字典形参 关键字实参数量无限

```
def 函数名(*args，参数名)
def 函数名(*，参数名)
```
命名关键字形参 必须是关键字实参

实参

```
函数名(数据1,数据2)
```
位置实参 按顺序对应

```
函数名(参数名1=数据,参数名2=数据)
```
默认形参 按名称对应

```
函数名(*序列)
```
序列实参 拆分 按顺序对应

```
def 函数名(**kwargs)
```
字典实参 拆分 按名称对应

5. 面向对象

创建类

```
class 类名:
    def __init__(self,参数):
        self.实例变量 = 参数
    def 实例方法(self,参数):
        操作实例变量
```

实例化对象

```
对象名 = 类名(数据)
```

访问实例成员

```
对象名.实例变量
对象名.实例方法(数据)
```

继承

```
class 子类名(父类名):
    def __init__(self,父类参数,子类参数):
        super().__init__(父类参数)
        self.子类实例变量 = 子类参数

儿子 = 子类名(父类数据,子类数据)
儿子.父类成员
儿子.子类成员
```

多态

```
class Model:
    def __str__(self):
        return 字符串
    def __eq__(self,other):
        return 相同的条件
    def __gt__(self,other):
        return 大小的条件
```

6. 高级语法

导入模块

```
import 模块名
模块名.成员

from 模块名 import 成员名
直接使用成员名
```

time模块

时间

```
时间戳 = time.time()
时间元组 = time.localtime()
```

转换

```
时间元组 = time.localtime(时间戳)
时间戳 = time.mktime(时间元组)
```

格式化

```
字符串 = time.strftime(格式, 时间元组)
时间元组time.strptime(字符串,格式)
```

异常处理

处理

```
try:
    可能出错的语句
except 异常类型:
    处理逻辑
except Exception:
    处理逻辑
finally:
    一定执行的逻辑
```

发送

```
raise 异常类型()
```

生成器

函数

定义

```
def 函数名():
    函数体
    yield 数据
```
返回多个结果

调用

```
生成器对象 = 函数名()    # 调用函数不执行
for item in 生成器对象:
```
省内存

```
生成器对象 =函数名()
容器类型(生成器对象)
```
转容器

内置生成器

```
for i,item in enumerate(容器):
```
读写容器元素

```
for item in zip(容器1,容器2)
```
合并对应元素

表达式

```
生成器对象 = (变量 for 变量 in 可迭代对象)
```

函数式编程

函数作为参数

高阶函数

```
def 通用函数(容器,条件):
    for item in 容器:
        if 条件(item):
            return item
```

lambda 表达式

```
结果 = 通用函数(容器,lambda 参数:函数体)
```

函数作为返回值

装饰器

增加新功能,不修改旧功能

```
def 装饰器名称(func):
    def wrapper(*args,**kwargs):
        新功能*
        result = func(*args,**kwargs)
        return result
    return wrapper
```
定义

```
@装饰器名称
def 旧功能():
```
使用

7. 文件操作

文件管理

创建路径

```
对象名 = Path(目录1,目录2)
```
相对路径

```
对象名 = Path.cwd().joinpath("目录1","目录2")
```
绝对路径

路径信息

```
对象名.absolute()  # 绝对路径(路径类型)
对象名.name  # 带后缀的完整文件名(str类型)
对象名.stem  # 文件名不带后缀(str类型)
对象名.suffix # 文件后缀(str类型)
对象名.parent  # 上一级路径(路径类型)
对象名.parts  # 分割路径(tuple类型)
对象名.exists()  # 路径是否存在(bool类型)
对象名.is_file()  # 是否文件(bool类型)
对象名.is_dir()  # 是否目录(bool类型)
对象名.is_absolute() # 是否绝对路径(bool类型)
对象名.stat().st_ctime  # 创建时间(时间戳)
对象名.stat().st_atime  # 访问时间(时间戳)
对象名.stat().st_mtime  # 修改的时间(时间戳)
对象名.stat().st_size  # 文件大小(字节Bete)
```

搜索目录

```
# 搜索当前目录所有路径(一层)
生成器 = 对象名.iterdir():

# 根据通配符搜索当前目录所有路径(一层)
生成器 = path.glob("通配符"):

# 根据通配符递归搜索当前目录所有路径(多层)
生成器 = 对象名.rglob("通配符")
```

新建路径

```
# 新建文件
对象名.touch()

# 新建目录
对象名.mkdir()
# 忽略目录存在时的报错
对象名.mkdir(exist_ok=True)
```

重命名

```
对象名.rename(新路径对象)
```

```
# 修改前：day03/homework/exercise01.py
old_file = Path("day03", "homework", "exercise01.py")
new_file_name = old_file.with_name("new_exercise.py")
# 修改后：day03/homework/new_exercise.py
old_file.rename(new_file_name)
```

删除路径

```
# 删除文件(永久删除，回收站不存在)
对象名.unlink()

# 删除目录(目录必须为空)
对象名.rmdir()

# 递归删除
shutil.rmtree(路径) # 永久删除，回收站不存在
```

拷贝

```
# 拷贝文件
shutil.copy(源文件,目标文件或目录)

# 拷贝目录(目标目录不能存在)
shutil.copytree(源目录,目标目录)
```

文件读写

文本文件

打开

```
with open(文件路径,"操作模式",encoding="编码方式") as 对象名:
    通过对象名操作文件
```

模式	含义	文件不存在时	是否清空文件	文件指针位置
r	只读	报错	不清空	开头
r+	读写			
w	只写	创建	清空	
w+	写读			
a	追加		不清空	结尾
a+	追加读			

操作模式

操作

```
# 读取指定数量字符
字符串 = 对象名.read(字符数)
列表 = 对象名.readlines()
# 读取文件每行字符
for 行 in 对象名:
```
读取

```
# 将字符串写入到文件
字符数 = 对象名.write(字符串)
```
写入

二进制文件

打开

```
with open(文件路径,"操作模式") as 对象名:
    通过对象名操作文件
```

模式	含义	文件不存在时	是否清空文件	文件指针位置
rb	只读	报错	不清空	开头
rb+	读写			
wb	只写	创建	清空	
wb+	写读			
ab	追加		不清空	结尾
ab+	追加读			

操作模式

操作

```
# 读取文件中指定数量字节
字节串 = 对象名.read(字节数)
```
读取

```
# 将字节串写入到文件
字节数 = 对象名.write(字节串)
```
写入