Python 速成

- 注释
- 数据类型 & 变量
- 运算符
- 条件语句
- 循环语句
- 函数
- 类和对象
- 文件操作
- 模块
- 异常处理

1. 注释

注释是代码中不被执行的说明文字，用于提高可读性。

- **单行注释**：用 `#` 开头，其后内容为注释

```
# 这是单行注释
print("Hello")  # 行尾也可以加注释
```

- **多行注释**：用三个单引号 `'''` 或三个双引号 `"""` 包裹（通常用于函数/类的文档说明）

```
'''
这是多行注释
可以写多行内容
'''
def add(a, b):
    """计算两个数的和"""
    return a + b
```

2. 数据类型 & 变量

变量

- 变量是存储数据的容器，无需声明类型（动态类型），直接赋值即可。
- 命名规则：只能包含字母、数字、下划线，不能以数字开头，区分大小写，不能用关键字（如 `if`、`for`）。

```
name = "Alice"  # 字符串变量
age = 18  # 整数变量
is_student = True  # 布尔变量
```

基本数据类型

类型	说明	示例
int	整数（正负整数、0）	100 、 -5 、 0
float	浮点数（小数）	3.14 、 -0.5
str	字符串（文本，单/双引号包裹）	"hello" 、 'Python'
bool	布尔值（真/假）	True （1）、 False （0）

复合数据类型

类型	说明	示例
list	列表（有序、可变、可重复）	[1, 2, "a"]
tuple	元组（有序、不可变、可重复）	(1, 2, "a")
dict	字典（键值对，无序，键唯一）	{"name": "Bob", "age": 20}
set	集合（无序、无重复、可变）	{1, 2, 3}

常用操作示例：

```python
# 列表（增删改查）
nums = [1, 2, 3]
nums.append(4)  # 增：[1,2,3,4]
nums[0] = 0     # 改：[0,2,3,4]
del nums[1]     # 删：[0,3,4]

# 字典（键值对操作）
person = {"name": "Alice", "age": 18}
print(person["name"])  # 查：Alice
person["gender"] = "女"  # 增：添加新键值对
person["age"] = 19      # 改：更新值

# 字符串（切片）
s = "Python"
print(s[0:3])  # 取前3个字符：Pyt（左闭右开）
```

3. 运算符

算术运算符

运算符	说明	示例
+	加	2 + 3 → 5
-	减	5 - 2 → 3
*	乘	2 * 3 → 6
/	除（浮点数）	5 / 2 → 2.5
//	整除（取整）	5 // 2 → 2
%	取余	5 % 2 → 1
**	幂运算	2**3 → 8

赋值运算符

= （基础赋值）、 += 、 -= 、 *= 等（简化运算+赋值）：

```python
a = 5
a += 3  # 等价于 a = a + 3 → a=8
```

比较运算符

结果为布尔值（ `True` / `False` ）：

`==` （等于）、 `!=` （不等于）、 `>` （大于）、 `<` （小于）、 `>=` （大于等于）、 `<=` （小于等于）

```
print(2 == 2)  # True
print(3 > 5)   # False
```

逻辑运算符

- `and` （与）：全为真则真
- `or` （或）：有一真则真
- `not` （非）：取反

```
print(True and False)  # False
print(3 > 2 or 1 > 5)  # True
print(not True)        # False
```

其他常用运算符

- 成员运算符： `in` （是否包含）、 `not in` （是否不包含）

  ```
  print(2 in [1,2,3])  # True
  ```

- 身份运算符： `is` （是否同一对象）、 `is not` （是否不同对象）

  ```
  a = [1]
  b = [1]
  print(a == b)  # True（值相等）
  print(a is b)  # False（不是同一对象）
  ```

4. 条件语句

根据条件执行不同代码块，用 `if` 、 `elif` （else if）、 `else` ，**缩进（4个空格）** 决定代码块范围。

```python
score = 85

if score >= 90:
    print("优秀")
elif score >= 80:  # 上一个条件不满足时判断
    print("良好")
elif score >= 60:
    print("及格")
else:  # 所有条件都不满足时执行
    print("不及格")  # 输出：良好
```

match - case 语句：

```python
def handle_data(data):
    """根据不同类型/结构的数据进行处理"""
    match data:
        # 1. 匹配具体值
        case 0:
            print("匹配到数字 0")

        # 2. 匹配类型（并绑定变量）+ 带条件判断（guard）
        case int(num) if num > 0:  # 匹配正整数
            print(f"匹配到正整数: {num}")
        case int(num) if num < 0:  # 匹配负整数
            print(f"匹配到负整数: {num}")

        # 3. 匹配字符串（支持部分匹配）
        case "hello":
            print("匹配到字符串 'hello'")
        case str(s) if s.startswith("error"):  # 匹配以 error 开头的字符串
            print(f"匹配到错误信息: {s}")

        # 4. 匹配列表/元组的结构（按长度和元素匹配）
        case [x, y]:  # 匹配长度为 2 的列表
            print(f"匹配到二元列表: 第一个元素 {x}, 第二个元素 {y}")
        case (a, b, c):  # 匹配长度为 3 的元组
            print(f"匹配到三元元组: 元素 {a}, {b}, {c}")

        # 5. 匹配字典（按键匹配，忽略额外键）
        case {"name": name, "age": age}:  # 匹配包含 name 和 age 键的字典
            print(f"匹配到用户信息: 姓名 {name}, 年龄 {age}")

        # 6. 通配符（匹配所有未被上面覆盖的情况）
        case _:
            print(f"未匹配到特定模式, 数据为: {data}")


# 测试不同类型的数据
handle_data(0)                  # 匹配到数字 0
handle_data(10)                 # 匹配到正整数: 10
handle_data(-5)                 # 匹配到负整数: -5
handle_data("hello")            # 匹配到字符串 'hello'
handle_data("error: 连接失败")   # 匹配到错误信息: error: 连接失败
handle_data([100, "apple"])     # 匹配到二元列表: 第一个元素 100, 第二个元素 apple
handle_data((True, 3.14, "ok")) # 匹配到三元元组: 元素 True, 3.14, ok
```

```
handle_data({"name": "Alice", "age": 30, "gender": "女"})  # 匹配到用户信息：姓名 Alice，年龄 30
handle_data(True)                # 未匹配到特定模式，数据为：True
```

- 关键特性说明：

1. **值匹配**：直接匹配具体的字面量（如 `case 0`、`case "hello"`）。
2. **类型匹配**：通过 `类型名(变量名)` 匹配特定类型（如 `case int(num)` 匹配整数并将值绑定到 `num`）。
3. **结构匹配**：支持列表、元组、字典的结构解析（如 `case [x, y]` 匹配长度为 2 的列表，`case {"name": name}` 匹配包含 `name` 键的字典）。
4. **条件判断（guard）**：通过 `if` 附加额外条件（如 `case int(num) if num > 0` 仅匹配正整数）。
5. **通配符 `_`**：用于匹配所有未被前面 `case` 覆盖的情况，类似 `default`。

5. 循环语句

for循环

遍历序列（列表、字符串、range等）：

```
# 遍历列表
fruits = ["苹果", "香蕉", "橘子"]
for fruit in fruits:
    print(fruit)  # 依次输出每个水果

# 配合range()生成数字序列（range(开始，结束，步长)，左闭右开）
for i in range(1, 5):  # 1,2,3,4
    print(i)
```

while循环

当条件为真时重复执行，需避免死循环：

```
count = 0
while count < 3:
    print("循环中")
    count += 1  # 必须更新条件，否则死循环
# 输出3次"循环中"
```

循环控制

- `break`：跳出当前循环

- `continue`：跳过本次循环，进入下一次

```
for i in range(5):
    if i == 2:
        continue  # 跳过i=2
    if i == 4:
        break     # 当i=4时跳出循环
    print(i)  # 输出：0,1,3
```

6. 函数

函数是可复用的代码块，用 `def` 定义，可接收参数并返回结果。

基本定义与调用

```
def greet(name):  # name是参数
    """打招呼的函数"""
    return f"Hello, {name}!"  # 返回值

# 调用函数
result = greet("Bob")
print(result)  # 输出：Hello, Bob!
```

参数类型

- **位置参数**：按顺序传递
- **关键字参数**：按参数名传递（可乱序）
- **默认参数**：定义时指定默认值（带有缺省参数的参数后面不能存在没有缺省参数的参数）
- **不定长参数**：`*args`（接收多个位置参数，存为元组）、`**kwargs`（接收多个关键字参数，存为字典）

```
def func(a, b=2, *args, **kwargs):  # b是默认参数
    print(a, b, args, kwargs)

func(1)  # 1 2 () {}（a=1，b用默认值）
func(1, 3, 4, 5, x=6, y=7)  # 1 3 (4,5) {'x':6, 'y':7}
```

** 匿名函数（lambda）**简化的单行函数，用 `lambda` 定义，只能有一个表达式：

```python
add = lambda x, y: x + y  # 等价于def add(x,y): return x+y
print(add(2, 3))  # 5
```

7. 类和对象面向对象编程（OOP）的核心，类是模板，对象是类的实例。

类的定义与对象创建

```python
class Person:  # 定义类（首字母通常大写）
    # 类属性（所有实例共享）
    species = "人类"

    # 构造方法：初始化实例时调用（self是当前实例本身）
    def __init__(self, name, age):
        # 实例属性（每个实例独有）
        self.name = name
        self.age = age

    # 实例方法（必须包含self参数）
    def greet(self):
        return f"我是{self.name}，今年{self.age}岁"

# 创建对象（实例化）
p1 = Person("Alice", 18)
print(p1.name)  # 访问实例属性：Alice
print(p1.greet())  # 调用实例方法：我是Alice，今年18岁
print(p1.species)  # 访问类属性：人类
```

继承子类继承父类的属性和方法，可扩展或重写：

```
class Student(Person):  # 继承Person类
    def __init__(self, name, age, school):
        # 调用父类构造方法
        super().__init__(name, age)
        self.school = school  # 新增子类属性

    # 重写父类方法
    def greet(self):
        return f"我是{self.name}，在{self.school}上学"

s1 = Student("Bob", 16, "一中")
print(s1.greet())  # 输出：我是Bob，在一中上学
```

8. 文件操作 用于读写本地文件，核心是 `open()` 函数，推荐用 `with` 语句自动关闭文件（避免资源泄露）。

模式	说明
r	只读（默认），文件不存在则报错
w	只写，文件不存在则创建，存在则覆盖
a	追加，文件不存在则创建，内容加在末尾
r+	读写
b	二进制模式（如 rb 、 wb ，用于非文本）

读写示例

```
# 写入文件（with语句自动关闭）
with open("test.txt", "w", encoding="utf-8") as f:
    f.write("Hello, Python!\n")  # 写入一行
    f.writelines(["第一行\n", "第二行\n"])  # 写入多行

# 读取文件
with open("test.txt", "r", encoding="utf-8") as f:
    print(f.read())  # 读全部内容
    # print(f.readline())  # 读一行
    # print(f.readlines())  # 读所有行，返回列表
```

9. 模块 模块是包含Python代码的文件（ .py ），用于代码复用；多个模块组成包（含 __init__.py 的文件夹）。

导入模块

```
# 1. 导入整个模块
import math
print(math.sqrt(4))  # 调用：模块名.函数 → 2.0

# 2. 导入模块中的特定内容
from math import sqrt, pi
print(sqrt(9))  # 直接用函数名 → 3.0
print(pi)  # 3.14159...

# 3. 给模块起别名
import datetime as dt
print(dt.datetime.now())  # 当前时间

# 4. 导入所有内容（不推荐，易冲突）
from math import *
```

自定义模块

创建 `mymodule.py`：

```
def add(a, b):
    return a + b
```

在同目录下的文件中使用：

```
import mymodule
print(mymodule.add(2, 3))  # 5
```

10. 异常处理

当代码出错时（如除以0、文件不存在），用 `try-except` 捕获异常，避免程序崩溃。

基本结构

```
try:
    # 可能出错的代码
    num = int(input("请输入数字："))
    print(10 / num)
except ValueError:  # 捕获特定异常（输入非数字）
    print("输入错误，请输入数字！")
except ZeroDivisionError:  # 捕获除以0的异常
    print("不能除以0！")
except Exception as e:  # 捕获其他所有异常（通用）
    print(f"出错了：{e}")
else:  # 无异常时执行
    print("执行成功")
finally:  # 无论是否异常都执行（如关闭资源）
    print("操作结束")
```

主动抛出异常

用 `raise` 手动触发异常：

```
def check_age(age):
    if age < 0:
        raise ValueError("年龄不能为负数")  # 抛出异常

try:
    check_age(-5)
except ValueError as e:
    print(e)  # 输出：年龄不能为负数
```