Python

一、 基础知识

1. 占位符

`%s` 字符串

`%d` 整数

`%f` 浮点数

2. 数字精度控制

“m.n”

m：数字的总宽度（位数）

n：小数部分的宽度（位数），并四舍五入

```
print(“ %7.2f ”, 11.345)
输出：_ _ 11.35
```

3. 字符串格式化

f “{占位}”

```
name = “zzp”
print(f “我是{name}”)
```

4. 类型转换/判断

```
int() float() str()
```

类型判断：

- `==` 用于比较对象的值是否相等。
- `is` 用于比较对象的身份标识是否相等（是否是同一个对象）。

5. input输入

input() 接收，默认存为str类型

6. 布尔类型bool

True = 1

False = 0

7. 循环

```
continue:
    终止本次循环，进行下一次
break:
    终止所有循环（跳出循环）
```

8. 异常处理

- **try-except**：用于捕获和处理代码块中的异常。**finally**：无论是否发生异常，都会执行的代码块。

```
try:
    可能发生错误的代码
except:
    如果发生异常就执行的代码
else：
    没有异常时执行的代码
finally：
    无论如何都会执行的代码
```

- **raise**：用于抛出自定义的异常。

9. 函数

- 函数可作为另一个函数的输入

```
def fun1(...)
    ...
def fun2(fun1)
    fun1(...)
```

- `lambda` 表达式：一个匿名的函数，只临时使用一次。

```
lambda 参数表：函数体
lambda x,y : x+y
```

return个yield

- `return` 用于从函数中返回一个值，并终止函数的执行。
- `yield` 用于生成一个迭代器，函数执行暂停并保存状态，可以在下一次调用中继续执行。函数的状态在多次调用中保持。

10. 深拷贝和浅拷贝

浅拷贝和深拷贝是关于复制对象的两个概念，它们的主要区别在于`复制的程度`。浅拷贝只复制对象的第一层，深拷贝复制对象的所有层。修改原始对象中的嵌套对象不会影响浅拷贝，但会影响深拷贝。

- 浅拷贝（Shallow Copy）：

`定义`：浅拷贝创建一个新对象，然后将原对象中的元素（对象的引用）复制到新对象中。新对象中的元素仍然是原对象中元素的引用。

`复制层次`：只复制了对象的一层。如果对象中包含引用类型的元素（例如列表或字典），则新对象中的引用仍然指向原对象中的相同元素。

`使用方法`：使用 copy() 函数或切片操作 [:] 来进行浅拷贝。

- 深拷贝（Deep Copy）：

`定义`：深拷贝创建一个新对象，并递归地复制原对象中的所有元素，包括嵌套的元素。新对象中的元素是原对象中元素的副本，而不是引用。

`复制层次`：复制了对象的所有层次结构，包括嵌套的对象。

`使用方法`：使用 copy.deepcopy() 函数来进行深拷贝。

- 拷贝的地址问题：

1. 在浅拷贝和深拷贝中，都会复制对象的地址，但它们对于嵌套的对象（即对象内部包含其他对象）的处理方式不同。
2. **浅拷贝：**复制对象的一层结构，但对于嵌套的对象，只复制其引用，而不是创建副本。因此，浅拷贝中的元素仍然指向原始对象中相同的地址。
3. **深拷贝：**递归地复制对象的所有层次结构，包括嵌套的对象。深拷贝中的元素是原始对象中元素的副本，而不是引用。

二、数据结构

1. 列表list

- **定义**：列表是`有序可变的元素集合`，用方括号包裹，元素之间用逗号`,`分隔。接近C++的`vector`容器

- **特性**：
  - 可以包含任意类型的数据，即列表中元素可以是不同类型。
  - 支持索引和切片操作。
  - 可以动态增删改查元素，支持append、extend、insert、remove、pop等方法。
  - 可以进行排序、反转、合并等多种操作。

`[元素1, 元素2, ...]`

索引： A[0]、A[0][1]

```
创建：mylist = []
查找：A.index(元素)
插入：A.insert(位置，元素)
增加：A.append()
删除：del A[1]
    A.pop(1)
    A.remove(元素)
清空：A.clear()
复制：list1[:] = list2
```

2. 元组tuple

B = tuple()，不可修改的列表

- **定义**：元组类似于列表，是`有序不可变的元素集合`，用圆括号`()`定义。
- **特性**：
  - 同样可以包含任意类型的数据。
  - 也支持索引和切片操作。
  - 一旦创建就不能修改（虽然可以通过重新赋值创建新的元组来达到替换的效果）。
  - 因为不可变性，元组在作为函数返回值或者在不希望数据被意外更改的情况下非常有用。

3. 字符串string

C = “zzp”

索引： C[2] = p

C.index(“p”) = 2

切片： `C[起:止:步长]` 注：止不包含

4. 字典dictionary

- **定义**：字典是一个`无序的键值对的集合`，`键唯一`的且不可变类型，值可以是任意类型。用花括号`{}`定义，键值之间用冒号`:`分隔，各个键值对之间用逗号`,`分隔。

- **特性**：
  - 键值对通过键来查找和访问，而不是通过位置索引。
  - 不保证键的顺序。
  - 支持添加、删除、修改和查找操作，通过 `dict[key]` 访问、 `del dict[key]` 删除、 `dict[key]=value` 修改键值对，`in` 关键字检查是否存在键。

```
D = {key1:value1, key2:value2, ...}
D[key1] = value1
```

5. 集合set

- **定义**：集合是一个 `无序且唯一元素的集合` ，用大括号 `{ }` 或者 `set()` 函数创建。
- **特性**：
  - 集合中的元素不允许重复，自动去重。
  - 不支持索引和切片操作，因为它没有固定顺序。
  - 提供交集、并集、差集等数学集合操作，如 `union` , `intersection` , `difference` 等。

6. 字节串与字节数组

`字节串bytes` ：用于表示二进制数据，字节的数量即为占用的字节数。如 `b = b'hello'` 。

`字节数组bytearray` ：可变的字节串，允许修改元素，与字节串类似，占用的字节数等于元素的数量。如 `ba = bytearray(b'hello')` 。

7. 装饰器

- `装饰器` 是一种用于修改函数或方法行为的高级技术。
- 装饰器可以在不改变函数代码和调用方式的情况下给函数添加新的功能，本质上是一个嵌套函数，接收被装饰的函数(func)作为参数，并返回一个包装过的函数，以实现不影响函数的情况下添加新的功能。
- 抽离出大量与函数主体功能无关的代码，增加一个函数的重用性。
- `应用场景` ：性能测试（统计程序运行时间）、插入日志、权限校验

三. 面向对象

1. 类(class)

类似于在线收集表

```
class Student:
    name = None
    age = None
```

```
    stu_1 = Student()
    stu_1.name = “zzp”
    stu_1.age = “23”
```

类包含属性和方法，类定义了一组属性（称为成员变量）和方法（称为成员函数）。对象是类的实例，是类定义的具体实现。

2. 类的方法

类内部的函数

def fun(self, 形参1, ...)

    ...

例：

```
class Student:
    name = None
    def fun1(self):
        print(f “我是{self.name}”)
    def fun2(self, meg):
        print(f “大家好，{msg}”)
stu_1 = Student()
stu_1.fun1()
stu_1.fun2(“我是zzp”)
```

3. 构造方法

```
class Student:
    def __init__(self, age, name):
        self.name = name
        self.age = age
        stu_1 = Student(“zzp”, 23)
```

注：类内部的属性/方法相互调用时，需在前面加self.

4. 封装（私有属性/方法 ）

前面加上两个下划线 `__` ，不可在类外调用，只能在类内调用。

5. 继承

```
class Phone:
    ...
class Newphone(Phone):
    ...
```

Newphone继承了Phone所有的属性和方法，在Newphone类中只需写新的属性和方法

6. 多继承

```
class Multiphone(Phone1, Phone2, ...):
    pass
```

7. 复写

继承后如果需改写一些属性/方法，只需重在子类新定义即可

`超类`：调用父类中的属性/方法

super()....

8. 多态

同样的函数，输入不同，输出不同。

例：

```python
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        print("汪汪汪")

class Cat(Animal):
    def speak(self):
        print("喵喵喵")
```

```
def make_noise(animal: Animal):
    animal.speak()

dog = Dog()
cat = Cat()

make_noise(dog)      # 输出：汪汪汪
make_noise(cat)      # 输出：喵喵喵
```

以父类做声明，以子类做实际工作

10. 注解

```
def func(形参名: 类型, 形参名: 类型, ...):
    pass
# 例子：
def add(x:int, y:int):
    return x+y
```

四、 MySQL

> ✅ **库-表-数据**

1. 打开：

Cmd输入 `mysql -uroot -p`

```
show databases； # 查看有哪些数据库
use 数据库名； # 使用某个数据库
show tables； # 查看数据库内有哪些表
exit； # 退出MySQL
```

2. SQL语言：

- 数据定义：DDL（Data Definition Language）
  - 库的创建删除、表的创建删除等
- 数据操纵：DML（Data Manipulation Language）
  - 新增数据、删除数据、修改数据等
- 数据控制：DCL（Data Control Language）
  - 新增用户、删除用户、密码修改、权限管理等
- 数据查询：DQL（Data Query Language）
  - 基于需求查询和计算数据

3. SQL注释：

- 单行注释： -- 注释内容（--后面一定要有一个空格）
- 单行注释：# 注释内容（# 后面可以不加空格，推荐加上）
- 多行注释：/* 注释内容 */

4. 数据定义DDL

库操作：

```
show databases; # 查看数据库
use 数据库名称; # 使用数据库
create database 数据库名称 [charset utf8]; # 创建数据库，[]表示可选
drop database 数据库名称; # 删除数据库
select database(); # 查看当前使用的数据库
```

表操作：

```
show tables; # 查看有哪些表
drop table [if exists]; # 删除表
# 创建表
create table student(
        id int,
        name varchar(10),
```

```
    age int
);
```

注： 列类型有：

·int 整数

·float 浮点数

·varchar(length) 文本，长度为length

·date 日期

·timestamp 时间戳

5. 数据操作DML

·表中数据插入

```
insert into 表(列1, 列2, ……) values(值1, 值2, ……), (值1, 值2, ……);
```

例：

```
CREATE TABLE student(
    id INT,
    name VARCHART(20),
    age INT
);

# 仅插入id列数据
INSERT INTO student(id) VALUES(10001), (10002), (10003)
# 插入全部列数据
INSERT INTO student(id, name, age) VALUES(10001, '周杰轮', 31), (10002, '王力鸿', 33),
(10003, '林俊节', 26)
# 插入全部列数据，快捷写法
INSERT INTO student VALUES(10001, '周杰轮', 31), (10002, '王力鸿', 33), (10003, '林俊节',
26)
```

·数据删除

```
delete from 表名 [where 条件判断];
delete from student where id = 1;
```

·数据更新

```
update 表名 set 列=值 [where 条件判断]
update student set name = 'dsm' where id > 2
```

7. 数据查询DQL

```
select from 字段列表/* 表 [where];
select id, name, age from student; select * from student;
```

8. 分组聚合与窗口函数


```
select gender, avg(age) from student group by gender ;
```

窗口函数：通常用于解决需要比较同一数据集中 不同行数据 的问题

假设有一个销售数据表sales，包含以下列：year（年份）、department（部门）、revenue（收入）。如果要计算每个部门每年的收入排名，可以使用 ROW_NUMBER() 窗口函数：

```
SELECT year, department, revenue,
ROW_NUMBER() OVER (PARTITION BY year, department ORDER BY revenue DESC) as rank
FROM sales;
```

PARTITION BY year, department 表示按年份和部门进行分区

ORDER BY revenue DESC 表示在每个分区内按收入降序排序

9. 排序

```
...order by...asc/desc 升序/降序
...limit n 限制前n条
...limit n, m 从n条开始，向后看m条
```

关键字顺序： from – where – group by/聚合函数 – select – order by – limit

10. 表的链接

Table1		
PersonID	PersonName	DepartID
1	张梓培	A
2	大司马	B
3	于龙	D

Table2	
DepartID	DepartName
A	研发
B	销售
C	HR

```
# 内连接：只返回两个表中匹配成功的记录
    SELECT Table1.PersonName, Table2.DepartName
    FROM Table1 INNER JOIN Table2
    ON Table1.DepartID = Table2.DepartID;
# 左连接：返回左表所有记录及右表中匹配的记录，如果没有匹配则包含NULL
    SELECT Table1.PersonName, Table2.DepartName
    FROM Table1 LEFT JOIN Table2
    ON Table1.DepartID = Table2.DepartID;
# 右连接：与左连接相反
# 全外连接：返回两表全部记录，缺少则补NULL
    左连接代码
    UNION
    右连接代码
```

内连接	
PersonName	DepartName
张梓培	研发
大司马	销售

左连接	
PersonName	DepartName
张梓培	研发
大司马	销售
于龙	NULL

三、Python操作MySQL

```
from pymysql import Connection
con = Connection(
    host='localhost', # 主机名（IP地址）
    port=3306, # 端口，默认3306
    user='root', # 账户名
    password='wu123n123c',
    autocommit = True # 自动提交
)
cursor = con.cursor() # 获取游标对象

con.select_db("test")
```

```
cursor.execute("create table test(id int)") # 创建表
con.commit()  # 确认修改，autocommit = True 时可以不写这一句
con.close()
```