# Python基础第一节

## 为什么运维中要用Python

### 什么是Python

Python是一种解释型、面向对象、动态数据类型的高级程序设计语言。它的设计理念强调代码可读性和简单性，使得开发者能够快速上手并高效完成任务，通常运用于机器学习、量化交易、运维、数据分析中，也有用python进行后端开发的岗位，包括在网络安全中，python都是一门极为重要的语言。

### Python在运维工作中的优点

Python自身的优点就是：简单、易学、速度快，开源、高层语言、可移植性、解释性、可扩展性、可嵌入性，丰富的库，独特的语法。它是一种胶水语言，可以把其他语言写的模块结合在一起。

而在系统运维上的优势在于其强大的开发多能力和完整的工业链，它的开发能力远强于各种shell和Perl，虽然shell脚本确实可以实现自动化运维，但由于shell本身的可编程能力较弱，对很多日常维护中需要的特性支持不够，也没有现成的库可以借鉴，各种功能都需要从头写起，所以说Python应该是每个运维工程师需要掌握的语言。

## Linux下安装Python

这里用Ubuntu为例：

首先先更新软件包数据库

```
sudo apt update
```

然后用apt包安装

```
sudo apt install python3
```

安装完成后，运行以下命令

```
python3 --version
```

如果出现版本号，说明安装成功

## Python语法基础

接下来我们就可以开始学习python的语法了，在这之前，就像其他语言一样，先让我们打出一句“Hello, World"

```
print("hello world")
```

可以看到，相比其他语言，python真的就是一句话的事。

现在，我们开始正式学习。

## 变量

变量是存储信息的容器。在其他语言，比如C中，你需要声明变量类型，但是在python中，它会在你赋值时自动确定。比如：

```
num = 10
name = "Nana"
is_female = True

#打印变量
print(num)
print(name)
print(is_female)
```

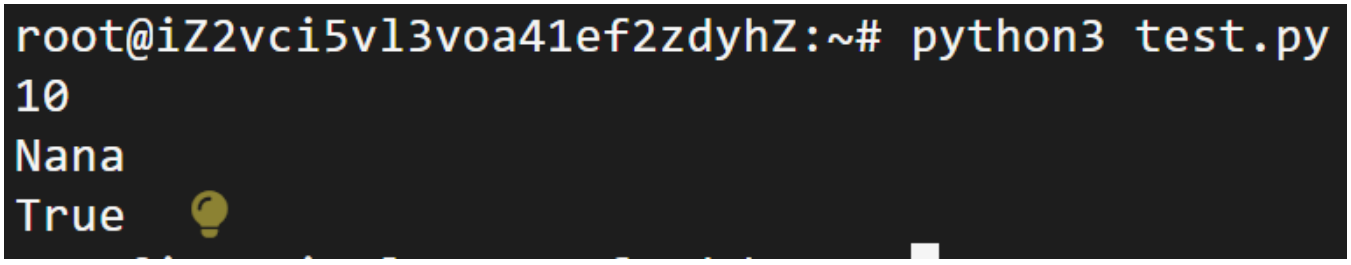
让我们测试一下结果，先创建一个文件：

```
vi test.py
```

写入内容，保存退出后运行：

```
python3 test.py
```

最后输出：




它自动识别了你是整数、字符串还是布尔类型

Python中的主要数据类型包括：整数（int）、浮点数（float）、字符串（str）和布尔值（bool）

布尔值代表真或假

## 算术运算符

```
a = 10
b = 3

print(a + b)  # 加法，输出：13
print(a - b)  # 减法，输出：7
print(a * b)  # 乘法，输出：30
print(a / b)  # 除法，输出：3.333...
print(a // b) # 整除，输出：3
print(a % b)  # 求余，输出：1
print(a ** b) # 幂运算，输出：1000
```

我们可以再测试一下：

```
root@iZ2vci5vl3voa41ef2zdyhZ:~# python3 test.py
13
7
30
3.3333333333333335
3
1
1000
```

### 比较运算符

```
x = 3
y = 10

print(x < y)
print(x > y)
print(x == y)
print(x != y)
print(x <= y)
print(x >= y)
```

大家动手试试看，输出结果是什么

### 逻辑运算符

```
x = True
y = False

print(x and y) #与运算
print(x or y) #或运算
print(not x) #非运算
```

```
root@iZ2vci5vl3voa3jx07vkbkZ:/home/admin# python3 test6.py
False
True
False
```

### 条件判断与循环控制

条件判断和循环结构是两个非常重要且常用的控制结构。它们允许你根据特定条件执行不同的代码块,以及重复执行某些代码块。

**条件判断**

if语句: 如果if后面的条件为True,则执行if下面的代码块。
elif语句: 如果前面的if条件为False,并且elif后面的条件为True,则执行elif下面的代码块。可以有多个elif语句。
else语句: 如果前面的所有if和elif条件都为False,则执行else下面的代码块。else语句是可选的。

经典例子:

```
score = 80

if score >= 90:
    print("优秀")
elif score >= 80:
    print("良好")
elif score >= 70:
    print("中等")
elif score >= 60:
    print("及格")
else:
    print("不及格")
```

**循环控制**

循环是重复执行某个代码块的结构，在python中，有for循环、while循环两种。

for循环用于遍历可迭代对象，如列表、元组、字符串等。在Python中，使用for关键字来实现for循环，后面跟的是一个可迭代对象和一个循环变量。

比如打印数字:

```
for i in range(10)
    print(i)
```

从0开始，到10结束，但是不包括10。

在实际开发中，range()函数可以用来生成各种整数序列，然后使用for循环来遍历这些序列，从而实现重复执行某个代码块的功能。

while循环在给定的条件为True时重复执行代码块。

break语句: 用于在循环中提前结束循环,即使循环条件还为True。

continue语句: 用于跳过当前循环的剩余部分,直接进入下一次循环。

```
i = 1
while i <= 10:
    if i == 5:
        break
    if i == 3:
        i += 1
        continue
    print(i)
    i += 1
```

（1）首先，将变量i赋值为1。

（2）使用while循环，判断i是否小于等于10，如果成立，则执行循环体内的代码。

（3）在循环体内，首先使用if条件判断，判断i是否等于5。如果i等于5，则执行break语句，退出当前循环。

（4）接着，使用另一个if条件判断，判断i是否等于3。如果i等于3，则执行i += 1语句，将i的值加1，接着执行continue语句，跳过本次循环的剩余代码，继续执行下一次循环。

（5）如果i不等于3，则执行print(i)语句，打印出i的值。

（6）最后，使用i += 1语句，将i的值加1，并继续判断循环条件，直到循环条件不成立，循环结束。

# 面向对象和类

面向对象(OO)是一种编程范式或思想,它将现实世界中的实体抽象为对象(Object),通过对象之间的交互来解决问题，在面向对象编程中,我们把数据和操作数据的方法封装在对象内部,通过对象的属性(Attribute)和方法(Method)来描述对象的特征和行为。

面向对象编程的主要特点包括:

对象(Object):对象是面向对象编程的基本单位,它代表现实世界中的实体或概念。每个对象都有自己的属性和方法,可以与其他对象进行交互。

类(Class):类是对象的蓝图或模板,定义了对象的属性和方法。通过类,我们可以创建出具有相同属性和方法的多个对象实例。

封装(Encapsulation):封装是指将数据和操作数据的方法绑定在一起,形成一个逻辑上的整体。通过封装,对象可以隐藏其内部细节,只暴露必要的接口给外部使用,提供了数据保护和隐藏。

继承(Inheritance):继承是一种机制,允许我们创建新的类(子类或派生类),这些类可以继承现有类(父类或基类)的属性和方法。通过继承,我们可以实现代码的重用和扩展。

多态(Polymorphism):多态是指同一操作或函数在不同的对象上可以具有不同的行为。通过多态,我们可以使用相同的接口来处理不同类型的对象,提高代码的灵活性和可扩展性。

Python提供了定义类和创建对象的语法,并支持面向对象的特性,如封装、继承和多态。

## 类的定义与封装

比如说，我们要描述一个人，ta的基本属性有：姓名、年龄、性别、身高，同时，他可以执行以下动作：说话、走路。如何用python的类来表示这个人呢

```
class Person:
    def __init__(self, name, age, gender, height):
        self.name = name
        self.age = age
        self.gender = gender
        self.height = height

    def speak(self, words):
        print(f"{self.name} says: {words}")

    def walk(self, distance):
        print(f"{self.name} walks {distance} meters.")
```

在这个Person类中,我们定义了四个属性:name,age,gender和height,它们分别表示一个人的姓名、年龄、性别和身高。这些属性在**init**方法中初始化。

我们还定义了两个方法:speak和walk,它们表示一个人说话、走路的动作。这些方法都接受一个参数,并在方法内部使用print函数输出相应的信息。

现在，让我们来学会调用它：

```
person = Person("Alice", 25, "Female", 165)

person.speak("Hello")
person.walk(100)
```

在这个例子中,我们创建了一个名为person的Person类实例,并传入了姓名、年龄、性别和身高的参数。然后,我们调用person实例的speak和walk方法,并传入相应的参数。我们可以看到,类可以用来描述现实生活中的事物,并将其属性和行为封装在一个单独的实体中。这种方式使我们能够以一种更加结构化和组织化的方式来表示复杂的系统和概念。

## 私有变量和公有变量

私有变量和公有变量的主要区别在于访问权限和使用场景。公有变量可以在类的内部和外部自由访问和修改，在python中，类的所有变量都默认为公有。公有变量的命名约定为小写字母和下划线组合，通常用于存储类的一般属性，这些属性可以被直接访问和修改，例如：

```
class Person:
    def __init__(self, name, age, gender, hight):
        self.name = name
        self.age = age
        self.gender = gender
        self.height = height

person = Person("Alice", 25, "Female", 165)
person.age = 26
print(person.age)
```

私有变量只能在类的内部访问和修改,外部代码无法直接访问或修改私有变量。在Python中,我们通过在变量名前面加上两个下划线__来表示一个变量是私有的。私有变量的命名约定是使用两个下划线作为前缀,后跟小写字母和下划线的组合，通常用于存储类的内部状态或敏感信息。例如：

```
class Account:
    def __init__(self, account_number, pay):
        self._account_number = account_number
        self._pay = pay
account = Account("12345", 100)
# account._pay = 200 这样赋值修改会报错，因为它已经是私有变量了
```

## 类的继承与多态

我们来通过继承创建一个新的类Student，它继承自Person类：

```
class Student(Person):
    def __init__(self, name, age, id):
        super().__init__(name, age)
        self.id = id

    def study(self, subject):
        print(f"{self.name} studies {subject}")
```

在这个例子中,Student类继承了Person类的所有属性和方法。通过使用super().**init**(name, age),我们调用了父类的构造函数,初始化了name和age属性。

# Python的常见内置函数

## 数值与数学函数

### abs(x)

返回数值的绝对值。可用于整数、浮点数或复数。

```
abs(-3)       # 3
abs(-3.5)     # 3.5
abs(3 + 4j)   # 5.0
```

### round(number[, ndigits])

返回 `number` 四舍五入的值。可选 `ndigits` 指定保留几位小数。

```
round(3.14159, 2)   # 3.14
round(3.5)          # 4
```

### hash(obj)

返回对象的哈希值，常用于哈希表、集合等

```
hash("abc")     # 整数，值视实现平台不同而不同
hash((1, 2, 3)) # 元组可哈希
```

### sum(iterable, /, start=0)

对 `iterable` 中所有元素求和，并加上 `start` 初始值（默认是 0）

```
sum([1, 2, 3])           # 6
sum((1.5, 2.5), 1.0)     # 5.0
```

## 类型转换函数

### float(x)

将字符串或整数转为浮点数

```
float("3.14")     # 3.14
float(10)         # 10.0
```

### str(obj)

将任意对象转换为字符串

```
str(123)          # '123'
str([1, 2])       # '[1, 2]'
```

## 字符串与字符处理函数

### ascii(obj)

返回对象的 ASCII 表示（转义非 ASCII 字符）

```
ascii("你好")       # '\u4f60\u597d'
```

### format(value, format_spec=”)

返回格式化字符串（等价于 `format()` 方法）

```
format(255, 'x')   # 'ff'
```

## 输入输出与执行控制

### input(prompt=None)

接收用户输入，返回字符串。

```
name = input("Your name: ")
```

## 其他常用函数

### open(file, mode='r', ...)

打开文件。

```
f = open("file.txt", "r")
```

### len(s)

返回长度

```
print(len("hello"))
```

内置函数是 Python 最精华的"基础工具库"，掌握它们，不仅能减少代码量、提升效率，还能**避免命名错误**和轮子重造

# 文件操作

## 文件的打开与关闭

```
# 打开文件（默认为只读模式）
file_path = 'example.txt'
with open(file_path, 'r') as file:
    # 执行文件操作，例如读取文件内容
    file_content = file.read()
    print(file_content)

# 文件在with块结束后会自动关闭，无需显式关闭文件
```

- `'example.txt'` 是文件的路径和名称，你可以根据实际情况修改为你想要打开的文件。
- `'r'` 表示只读模式。如果你想要写入文件，可以使用 `'w'` 模式，如果想要追加内容，可以使用 `'a'` 模式等
- `with open(...) as file` : 是使用上下文管理器的方式，确保文件在使用后被正确关闭，即使在处理文件时发生异常也能保证关闭(其实还有open()的写法，但是with open显然更好)

如果用open()的话，代码最后要用close()关闭文件，但是用with open()就不需要。

**2. 访问模式及说明**

| 访问模式 | 说明 |
| --- | --- |
| r | 以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。 |
| w | 打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。 |
| a | 打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |
| rb | 以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。 |
| wb | 以二进制格式打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。 |
| ab | 以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |
| r+ | 打开一个文件用于读写。文件指针将会放在文件的开头。 |
| w+ | 打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。 |
| a+ | 打开一个文件用于读写，如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果改文件不存在，创建新文件用于读写。 |
| rb+ | 以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头 |
| wb+ | 以二进制格式打开一个文件用于读写。如果改文件已存在则会覆盖。如果改文件不存在，创建新文件。 |
| ab+ | 以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果改文件不存在，创建新文件用于读写。 |

# 文件读写

## 写文件

### 写入文本文件

使用内置的 `open` 函数来打开文件并写入内容。确保使用适当的模式（例如，`'w'` 表示写入）

```
file_path = 'example.txt'

# 写入文件
with open(file_path, 'w') as file:
    file.write("Hello, how are you.")
```

## 读取文本文件

使用内置的 `open` 函数来打开文件并读取内容

```
file_path = 'example.txt'

# 读取文件
with open(file_path, 'r') as file:
    data = file.read()
    print(data)
```

## 逐行读取数据

`readlines` 是 Python 中用于读取文件的方法之一，它用于逐行读取文件内容，并将每一行作为字符串存储在一个列表中。下面是对 `readlines` 方法的详细解释：

```
with open('file.txt', 'r') as file:
    lines = file.readlines()
```

读数据（readline）
readline 是 Python 中用于读取文件的方法之一，它用于逐行读取文件内容，并返回文件中的一行作为字符串。以下是对 readline 方法的详细解释：
使用 readline 方法的基本语法

```
with open('file.txt', 'r') as file:
    line = file.readline()
```

### readlines 和 readline的区别

readlines 和 readline 是 Python 中用于读取文件的两种不同方法，它们之间有一些重要的区别：

**readlines 方法：**

返回类型：readlines 方法返回一个包含文件所有行的列表，其中每个元素都是文件中的一行文本字符串。

使用情况： 适用于处理包含多行文本的文件，可以一次性将整个文件加载到内存中。这种方法适用于文件较小，可以完全装入内存的情况。

**readline 方法：**

返回类型： readline 方法每次调用只返回文件中的一行作为字符串。如果再次调用，将返回下一行。当文件读取完毕后，返回空字符串 ''。

使用情况： 适用于逐行处理大型文件，可以有效地降低内存使用。因为它一次只读取一行，可以在循环中逐行处理文件，而不必将整个文件加载到内存中。

**区别总结：**

readlines 一次性读取整个文件的所有行，并返回一个包含所有行的列表。

readline 逐行读取文件，每次调用返回文件中的一行，适用于处理大型文件，减少内存占用。

readlines 返回包含换行符的每一行，而 readline 返回单独的行，需要手动去除换行符。

选择使用哪个方法取决于文件的大小和处理需求。如果文件较小，可以完全装入内存，使用 readlines；如果文件较大，可以逐行处理，使用 readline

（ps:这节课我们讲的文件操作都是针对文本文件，之后的进阶课我们会讲解模块的概念，python的强大之处才会真正体现出来，因为python的一些模块，我们可以更加快速的写爬虫或者处理数据、将数据生成直观的图片等等）

# 作业

**level0**

安装好python，打印“hello world"

**level1**

用课堂上讲过的面向对象和类的知识，打印出这样一句话：

Nana says: Hello, how are you?

Nana sings Love Like Fire song.

Nana likes Zhangbing

**level2**

给定一个数组nums，编写一个函数将所有0移动到数组的末尾，同时保持非零元素的相对顺序。

**level3**

输入成绩，先判断是否为合理数字（指不小于0不大于100），再判断等级区间.

A:【90，100】

B:【80，90)

C:【70，80)

D:【60，70)

F:【0，60)

**level4**

打开一个文件，写入几行诗，要求逐行读取

**level5【选做】**

生成随机数，写一个猜数小游戏。

完成后截图发：shizhan@lanshan.email，记得备注自己的姓名。