1. 常量和变量

常量：在程序中固定不变的值,一旦初始化后值就不会改变,比如数字常量 3.14,100,字符常量“test”,”wolf”,”python”,通常情况下,常量值的就是我们创建的对象

1.1. 变量的定义

变量: 是指一个可以存储数据的存储空间,对于具体存放什么样的数据没有规定,简单的讲,变量就是可以一个存放东西的容器,一般具有以下几个特点:

① 变量必须先赋值才可以使用 a = 100

② 如果变量出现在表达式中,那么参与运算的是变量中所存放的数据

③ 对于变量的名称我们称之为变量名,其命名规则符合标识符的特点

④ 变量名称区分大小写

⑤ 类型是输入对象的,而对于变量则是没有类型的概念,可以通过 type(变量名)查看其应用的数据类型

```
# 定义一个变量 a 并且赋值为 100
a = 100
# 输出 100
print(a)
# 报错 变量 b 没有定义 变量必须赋值以后才可以正常调用
# print(b)
# 定义一个新的变量 A 其中 a 和 A 是两个不同的变量,区分大小写
A = 200
# 同时打印多个值 a,A 使用逗号隔开
print(a,A)
# 变量本身是没有类型的,但是变量中的对象是有类型的
print(type(a)) # 数字类型 int
# 创建一个字符串对象"name" 并且把 name 对象的应用 赋值给变量 a
a ="name"
print(a) # 输出 name
```

1.2. 变量和对象的引用

对于对象(常量) 创建出来是存放在内存中,

变量中保存的并不是对象本身,而是指向对象的一个内存地址,即表示变量保存的是对象的引用

a = 100

①创建一个对象,值为 100

②创建一个变量 a

③让变量 a 指向对象 100

对于变量中所存放的引用的值可以通过 id(变量名) 来查看对象的内存地址值

1.3. 对象的垃圾回收

当一个变量 a 重新赋值后，那么原来的一个对象是没有变量引用的,如果一个对象没有被引用,那么会被当做垃圾被回收

a = 100

a = 'python'

如果一个对象没有被引用,没有变量指向该对象,那么该对象会被当成垃圾被回收,释放内存

1.4. 对象的共享引用

所谓变量的共享引用,是指把一个变量 a 赋值给另外一个变量 b,这时候变量 a 和变量 b 同时指向同一个对象,那么变量 a 和变量 b 是对该对象的共享引用

a = 100

b = a

```
# 创建一个字符串对象 jack 并且把 jack 的内存地址值赋值给 a
a = "jack"
# 变量 b 也指向 jack,即两个变量指向同一个对象
b = a
# 打印的结果是一样的
print(id(a),id(b))
a = "marry"
# 结果是 "marry","jack"
print(a,b)
```

2. 常用数据类型

2.1. 数字类型

数字类型主要分为整数(int),小数(float),复数,其中数据属于不可变的对象

主要注意的几点:

① 对于整数的表示的最大值没有限制,和系统的内存有关系

② 对于数据的表示方法可以使用十进制,十六进制,八进制,二进制

③ 可以通过 int()构造整数,通过 float()构造小数

④ 各个进制之间可以通过对应的方法进行转换 hex(),oct(),bin(),int(),float()

2.1.1. 数字的表示形式

```
# 十进制数据
num1 = 1001
```

```
# 十六进制数据
num2 = 0xff
# 八进制数据
num3 = 0o77
# 二进制数据
num4 = 0b1010
# 通过 print 打印的默认是十进制数据
print(num1,num2,num3,num4)
```

2.1.2. 表达式操作符

对于数字的基本的算术运算有 + - * / // % ** 等运算

x/y: 表示 x/y,有小数部分

x//y:表示 x 除以 y 取整数(向下取整)

x%y: 计算 x 除以 y 的余数

x**y: 计算 x 的 y 次方

```
x = 10
y = 3
# 3.3333
print(x/y)
# 3
print(x//y)
# 余数为 1
print(x%y)
# 1000
print(x**y)
```

2.1.3. 内置数学函数

pow(x,y) x 的 y 次方 和 x**y 等价

abs(x) x 的绝对值

int(x,type) 把 x(数字或者字符串)转换以一个 int 整数,type 用来指定进制,默认为十进制

hex(x) 把数字 x 转换为十六进制

oct(x) 把数字 x 转换为八进制

bin(x) 把数字 x 转换为二进制

```
print(pow(2,3))# 8
print(abs(-19))# 19
print(int("100"))# 100
print(int("100",2))# 4
print(hex(1024))# 0x400
print(oct(127))# 0o177
```

2.2. 布尔类型

Python3 中布尔类型(bool)只有两个值:True 和 False

布尔类型是数字类型的一个子集,可以和数字之间直接进行运算

① 把布尔类型转换为数字类型 True ==> 1 ,False ==> 0

② 把数字类型转换为布尔类型(bool(数字)) 0 ===>False 非 0 转换为 ===>True

③ 所有类型的对象都可以表示为 True 或者 False 可以通过 bool(对象) 来判断器表示的是 True 还是 False

```
flag = True
print(flag)
# True --> 1
num = flag +3
print(num) # 4
# 把字符串转换为 bool
# 空字符 代表 False,其他 True
print(bool("")) # False
print(bool("wolfcode"))# True
print(bool(0))# False
print(bool(-100))# True
```

2.2.1. random

random 是一个可以用来生成随机数的一个模块,使用之前需要先通过 import random 导入模块

通过 randint(start,end)可以生成 star 到 end 之间的随机整数(包括 start,end)

```
# 导入 random 模块
import random
# 打印 1 -10 之间的一个随机整数
print(random.randint(1,10))
```

2.3. 字符串类型

2.3.1. 字符串的表示方法

字符串是一个有序的不可变的一个字符的集合,用于存储和表现基于文本的信息

① 使用单引号

② 使用双引号

③ 使用三重引号: 如果需要使用多行表示字符串的时候可以使用三重引号

```
# 使用双引号
text = "wolfcode"
# 使用单引号 一定要注意引号需要匹配
print('你好')
# 使用三重引号
text2 = """ 这里的文字可以多行表示
```

```
好好学习
天天向上
"""
print(text2)
```

字符串的特点:

① 有序的序列, 可以通过索引找到具体的每一个字符

② 不可变的序列

③ 可以通过 len()获取字符的个数(字符串的长度)

④ 字符串的 + * 操作

```
# 使用双引号
text = "wolfcode"
# 通过索引找到某一个字符
# 第一个字符
print(text[0])
# 最后一个字符
print(text[-1])
# 获取到字符的长度  索引值的范围是 [0,len(text)-1]
print(len(text))
# 字符串连接一起,返回一个新的字符串
print("wolf"+"code")
# 输出多个相同的字符
print("="*40)
```

2.3.2. 转义字符

转义序列可以让我们在在字符串中表示不容易通过键盘输入或者是在输入过程中字符本省有一些特殊意义的,我们需要使用转义字符来表示,

转义字符的表示方法: 使用一个反斜杠+字符

比如 \' 这表示一个单引号' 而不在把单引号'看作为字符串的边界符

如果在\后面没有一个合法的转义字符,那么在字符串中就是一个反斜杠

转义字符是固定的几个,常用的

\r 返回 返回到当前行的开始位置

\\ 反斜杠,不在当做转义字符处理

\' 单引号,不会当成字符串的开始或者结束标志

\t 制表符,用于对齐数据格式

\n 换行符,新起一行

```
a = "\name\text"
# 返回 8
```

```
print(len(a))
# 有一个换行符,一个制表符 两个转义字符
print(a)
# 中间的双引号不会作为字符串的边界符
a = "ab\"c"
```

2.3.3. 原始字符串

如果一个字符串中含有\+特殊字符的话,那么会当成一个转义字符处理,如果不想当成转义字符,有两种方法解决

① 把反斜杠就当成普通字符处理 \\

② 在字符串前面加上字符 r 表示该字符中的所有\都是不同字符,不在具有反斜杠的意义

text = r"d:\name\test\read.txt"

```
# 原始字符串 使用 r 开头
filename = r"d:\test\name.txt"
print(filename)
# 两种方式的效果一样
filename2 ="d:\\test\\name.txt"
print(filename2)
```

2.3.4. 字符串的相互转换

int(): 把一个字符串转换为一个数字

str():把一个数据转换为字符串

```
print(str(123))
print(int("123"))
print(123+"456")
```

2.3.5. 字符串代码转换

ord():获取到字符对应的编码值(转换为 Unicode 的编码的值)

chr(): 获取到一个值对应的字符

```
# 需要注意的是只能转换单个字符
print(hex(ord('中')))
print(chr(20013))
```

2.3.6. 索引和分片

字符串是一个有序的集合,即表示字符串中的每个字符都是有顺序的,比如 "ab" 和 "ba"是两个不同的字符串

对于字符在字符串中的位置,我们使用索引来表示

切片是指对操作的对象截取其中一部分的操作.可以对字符串,列表,元组进行切片操作

切片的基本语法:

[起始:结束:步长]

操作的区间使用的是前闭后开的原则,包括起始索引,不包括结束索引,

如果步长不写,默认为 1

如果结束索引没有,默认为直接到最后一个元素

如果起始索引没有写,默认为 0

偏移操作: 正偏移:从左至右的偏移(偏移 0 位第一个元素)

负偏移: 从右至左的偏移(偏移-1 为最后一个元素)

```
# 对于有序的字符串对象,其中从左到右每个字符都有一个索引 从[0,len(x)-1]
text = "wolfcode"
#第一个字符
print(text[0])
# 最后一个字符
print(text[len(text)-1])
# 从右往左表示 依次为-1 -2 ... -len(text)
print(text[-1])
print(text[-8])
```

```
# 通过索引我们可以从字符串中获取到一个字符
# 如果需要从字符串中获取多个字符,需要使用切片,格式为 text[start:end] 不包括 end
text = "wolfcode"
# 需要 wolf 字符串,需要注意的是切片会返回一个新的字符串,不会改变原来的字符串
text2 = text[0:4] # wolfcode
# 如果开始索引是 0 可以省略: 前面的开始索引
text2 = text[:8] # wolfcode
# 如果需要截取到最后一个元素,可以省略: 后面的结束索引
text2 = text[1:]# olfcode
# 如果同时省略开始索引和结束索引,那么就相当于对原来的字符串的拷贝
text2 = text[:] # wolfcode
# 对于索引值,我们也可以使用负数表示
text2 = text[0:-1]  # wolfcod
# 如果需要获取 0 2 4 6 8 等偶数索引,我们可以在左切片的时候设定一个步长,默认为 1
text2 = text[::2] # wlcd
# 如果需要从后面往前截取字符串
text2 = text[-1::-2] # eofo
# 把一个字符串 反序打印
print(text[-1::-1]) # edocflow
print(text2)
```

语法格式:

text[start,end,step]

:start: 开始索引 如果不写, 默认为 0

end: 结束索引 如果不写, 默认为 len(text)

step: 步长,如果为正, 从左往右, 如果为负 则是从右往左

其中不包括结束索引,

对于上面的 start,end,step 的值,都可以设置为负数

2.3.7. 格式化字符串

所谓格式字符串,就是使用一个通用的模板来和一些对应的占位符来表示一个字符串,对于占位符可以使用相对应的字段替换

其中格式字符串有两种方式:

① 使用%占位符的方式

info="my name is %s and age is %s"%("jack",18)

② 使用 format 方法

info="my name is {0} and age is {1} ".format('jack,21')

```
# 使用%站位符号的方式
print("my name is %s, I am %d"%("叩丁狼",18))
text = "your name is %s, your job is %s"%("wolfcode","teach")
print(text)
# 使用 format 函数 在{}省略没写,相当于使用位置占位 0,1 依次占位
text2 ="your name is {1}, your job is {1}".format("wolfcode","teach")
print(text2)
# 使用名称占位符,对于后面的值也是需要指定名称来替换
text3 ="your name is {name}, your job is {job}".format(name="wolfcode",job="teach")
print(text3)
```

2.3.8. 常用方法

x.find(y): 返回 y 在 x 中第一次出现索引位置,如果没有找到,返回-1

x.index(y):返回 y 在 x 中第一次出现的索引位置,如果没有找到,返回一个错误 ValueError

x.repalce(y,z):在 x 中把 y 替换为 z,注意,是返回一个新的字符串,原来的字符串 x 的值不会改变

x.split([y]): 把 x 字符串按照 y 进行分割,如果不写,则使用空白字符,包括换行符,制表符等

y.join(z): 使用 y 字符把 z 中的数据依次连接起来

x.strip(): 返回一个去掉字符串 x 两边的空格的新字符串

x.encode([charset]) : 把字符串 x 编码为字节数据,其中 charset 如果不写,默认为 utf-8,如果需要把字节数据解码为字符串,需要使用 decode 方法

```
# 返回 code 在 wolfcode 中的索引 4
print("wolfcode".find("code"))
# 和 find 一样,如果找不到,报错 ValueError
print("wolfcode".index("code"))
text = "wolfcode"
# 替换字符串
print(text.replace("o","ooxx"))
# 原来的字符串不会改变
print(text)
text2 = "你好,谢谢,请,对不起,再见"
# 使用逗号分隔,返回多个数据(列表)
text3 = text2.split(",")
print(text3)
# 使用-连接 text3 中的每个元素
print("-".join(text3))
# encode 编码操作 如果不写编码规则,是用 utf-8,GBK,GB2312,如果只有英文字母,可以使用 ASCII
# 字节数据 b 开头
data = text2.encode("utf-8")
print(data)
# 解码,decode,需要注意的是解码的规则必须和编码使用的规则一致,否则会出现乱码的情况
print(data.decode("utf-8"))
```

3. 运算符操作

3.1. 关系运算符

```
"""
关系运算符 表达式返回一个布尔类型的值,True 或者 False
x==y 等于 比较两个对象的值是否相等 并且需要对象类型一样
x!=y 不等于 比较两个对象的值是否不相等
x>y 如果 x>y 返回 True,否则返回 False
x>=y
x<=y
"""
print(100==200) #False
print(100=="100") # False
print(1==True) # True 布尔类型也是属于整数类型的
print("bdc" > "azbbd")# 对于字符串的比较大小, 从左往右,依次比较每个字符的大小(Unicode 值)
```

3.2. 逻辑运算符

```
"""
关系运算符
逻辑与 and
x and y 如果 x 对应的布尔值为 False, 则返回 x,否则返回 y
```

```
逻辑或 or
x or y 如果 x 对应的布尔值为 True,则返回 x,否则返回 y
逻辑非 not
not x 如果 x 对应的布尔值为 True,则返回 False,否则返回 True
对于关系运算符具有短路效果,如果已经可以确定结果,那么就不会再去执行后面的表达式
"""
print(1 and 2 )# 2
print(False and True) # False
print(0 and True) # 0
print(1 or 2 )# 1
print(False or True) # True
print(0 or True) # True
print(not 0) # True
print(not "")# True
# 字符串和有空白字符是不一样的概念, 长度为 0 的字符称之为空字符
print(not " ")# false
```

3.3. 成员关系

```
"""
对象实体测试
x in y: 判断在 y 中是否包含 x,如果包含,返回 True,否则返回 False
x not in y: 判断在 y 中是否不包含 x,如果不包含,返回 True,否则返回 False
"""
x = "wolf"
y ="wolfcode"
print(x in y) # True
y ="coding"
print(x in y) # False
print(x not in y) #True
```

3.4. 实体对象测试

```
"""
对象实体测试
x is y: 用于判断 x 和 y 的引用是否一样,如果一样,返回 True,否则,返回 False
x is not y: 用于判断 x 和 y 的引用是否不一样,如果不一样,返回 True,否则,返回 False
对于成员关系 如果 x is y ==> id(x) == id(y) 如果 x is not y ==> id(x) != id(y)
"""
x = 1000
y = 1000
print(id(x),id(y))
print(x is y) #True
y = x
print(x is y ) #True
# 因为对于小的一些数据有缓存的原因
```

```
x = 10
y = 10
print(x is y) #True
```

4. 流程控制

4.1. 顺序结构

程序代码从上到下依次执行,Python 的解释器从上往下依次解释执行代码

```
"""
顺序结构
"""
name = "wolfcode"
print(name)
age = 18
print("name is {},age is {}".format(name,age))
```

4.2. 分支结构

对于代码执行有的时候并不是一直都是从上往下顺序执行的,需要进行条件判断才可以继续往下执行,比如说去商场买东西的时候,只有付钱了才可以把商品从商场拿走,这种结构就需要使用到逻辑判断的分支结构.在 Python 中,我们使用的是 if 语句完成逻辑判断

4.2.1. if 结构

```
"""
分支结构
"""
# is_page 是否付钱
is_page = True
print("---start----")
"""
    if 条件语法格式:
    if 表达式:
        执行语句 1
        执行语句 2
    执行语句 3
    我们把整个 if 语句称之为复合语句
"""
if is_page:
    # 缩进语句
    print("111")
    print("您的商品已经结账,欢迎下次光临")
print("---end----")
```

4.2.2. if else

```
"""
如果你的订单金额超过 128 直接 EMS 包邮
否则，韵达包邮
"""
amount = int(input("请输入您的订单金额"))
if amount>=128:
    print("金额超过 128")
    print("EMS 包邮")
else:# 不满足前面的条件,就执行 else
    print("韵达包邮")
print("---end---")
```

4.2.3. if elif...else

```
"""
if 结构 只需要处理一种情况的时候
if else 结构 处理两种情况
需求:
如果你的订单金额超过 128 直接 EMS 包邮
如果你的订单金额超过 68 直接韵达包邮
否则，邮费自理
if 表达式 1:
    执行语句 1
elif 表达式 2
    执行语句 2
else:
    执行语句 3
"""
amount = int(input("请输入订单金额"))
if amount >= 128:
    print("EMS 包邮")
elif amount >= 68:
    print("韵达包邮")
else:
    print("邮费自理")
```

4.2.4. if 嵌套

```
"""
if 语句嵌套
需求：你需要给你女朋友转账
```

```
    ① 判断你的卡的余额 如果需求小于你女朋友需要的钱,不能转账
    ② 如果你的卡的语句可以满足你女朋友的需要,
        ③ 判断你输入的密码是否正确,如果密码不正确
            提示密码错误
           否则 转账成功
"""
amount = 6000
use_amount = int(input("请告诉我你需要多少钱"))
if amount >= use_amount:
    password = input("请输入密码")
    if password == "123456":
        print("转账成功")
    else:
        print("密码不正确")
else:
    print("余额不足")
```

4.3. 循环结构

如果有一些代码需要重复不断的执行知道某一条件不满足的时候才不执行的代码,我们通常可以考虑使用循环结构来解决,编写通用的循环使用 while 语句,如果是对序列的循环遍历,我们可以只用 for 循环

4.3.1. while 循环

```
"""
老师需要让你数 1-100 的数字
while 循环格式:
while 表达式:
    语句 1
    语句 2
语句 3
"""
start = 1
print("---start---")
while start <= 10:
    print(start)
    start += 1
print("---end---")
```

```
"""
需求: 求 1-100 之间的和
"""
sum = 0
start = 1
```

```
while start <= 100:
    sum += start
    start += 1
print("1-100 的和是%d"%sum)
```

4.3.2. while 练习

打印矩形

```
"""
*****
*****
*****
"""
line = 1
while line <= 3:
    i = 1
    while i <= 3:
        print("*", end='')
        i += 1
    print()
    line +=1
```

打印三角形

```
"""
*
**
***
"""
line = 1
while line <= 3:
    i = 1
    while i <= line:
        print("*", end='')
        i += 1
    print()
    line +=1
```

打印九九乘法表

```
"""
1*1=1
1*2=2   2*2=4
1*3=3   2*3=6   3*3=9
"""
line = 1
```

```
while line <= 9:
    i = 1
    while i <= line:
        print("%d*%d=%d"%(i,line,i*line), end='\t')
        i += 1
    print()
    line +=1
```

4.3.3. for 结构

for 循环主要是用来遍历序列对象(字符串,列表,元组,集合等)

```
name = "wolfcode"
"""
index = 0
length = len(name)
while index < length:
    print(name[index])
    index += 1
"""
print("---start---")
# 通过 for 循环遍历方便,但是没法获取到对应的索引
for ch in name:
    print(ch)
print("---end---")
```

4.3.4. 循环控制

break 用于结束当前循环

continue 用于结束本次循环,继续下一次循环

```
"""
break 用于结束当前循环
continue 用于结束本次循环,继续下一次循环
1 打印 100 以内 最大的 5 个奇数的和
2 打印 100 以内,能被 3 整除的数的和
"""
num = 100
sum = 0
count = 0
while num >= 0 :
    if num%2 == 1:
        print("计算的基数为:%d"%num)
        sum+=num
        count += 1
        if count ==5:
```

```
            break
    num -=1
print("总和为%d"%sum)
```

```
sum = 0
num = 0
while num <=100:
    num +=1
    if num%3!=0:
        continue
    sum +=num
print("总和",sum)
```