Shell, Python 编程基础

吴连近（高能所）

2020-10-1. 入门培训. 南华大学

Shell 基础

Hello World

- Shell（壳）提供使用者使用界面的软件，也就是命令解释器。与 Kernel, GUI 的关系。
- Shell 常用的类型 Bourne Again shell (bash), C shell (csh, tcsh), Korn shell (ksh), Bourne shell (sh)。查看 Shell 类型：

```
1 echo $SHELL
```

- 从 Hello World 开始。vi 创建文件，命名：helloworld.sh

```
1 #!/usr/bin/env bash #env是一个可执行命令，指定脚本调用合适的解释器（bash）执行
2 echo "Hello World"  #输出 Hello World
```

- 运行方式有多种：

```
1 chmod +x helloworld.sh
2 ./helloworld.sh
```

```
1 bash helloworld.sh
```

```
1 source helloworld.sh
```

```
1 sh helloworld.sh
```

Shell 变量和数据类型

- 变量的定义：var=value。（注意：“=”两边不能有空格）
- 变量名规则：
  只能用英文，数字，下划线，首个字符不能是数字
  中间不能有空格，可是用下划线
  不能使用标点符号
  不能使用 bash 关键字
- 变量名的使用：${变量名}

```
1 #!/usr/bin/env bash #env是一个可执行命令，指定脚本调用合适的解释器（bash）执行
2 var="Hello World" #定义变量var，并赋值为字符串"Hello World"
3 echo ${var}  #输出 Hello World
```

- 数据类型：
  字符串：

```
1 var="hello world"
```

  整数型：

```
1 var=2
```

  数组型：

```
1 var=(1 2 3 4 5)
```

均可以通过 declare 定义

字符串常用基本操作

- 字符串相加

```
var0="Hello"
var1="World"
var="${var0} ${var1}" #完成两个字符串的相加
echo ${var} #输出Hello World
```

- 字符串截取和替换 (expr,cut,awk,sed)

```
var="Hello_World_Welcome"

#截取前五个字符的方法
expr substr "${var}" 1 5
echo ${var} | cut -c 1-5
echo ${var} | awk '{print substr(, 1, 5)}'

#按照指定要求"_"分割提取第二列
echo ${var} | cut -d "_" -f 2
echo ${var} | awk -F "_" '{print $2}'

#字符串替换 将Hello替换为Hi
echo ${var} | sed 's/Hello/Hi/g'
echo ${var} | awk '{gsub(/Hello/, "Hi"); print $0}'
```

字符串本身携带诸多很秀的操作，这里不做具体介绍。熟悉 expr,cut,sed,awk 等命令，可以完成更多更复杂的操作

整数型常用基本操作

- 整数型计算

```
1  var0=1
2  var1=2
3
4  #方法1
5  var=$((${var0}+${var1}))
6
7  #方法2
8  var=$[$(var0) + ${var1}]
9
10 #方法3
11 var=‘expr ${var0}+${var1}‘
12
13 #方法4
14 var=‘echo "${var0}+${var1}"|bc‘
15
16 #方法5
17 let var=${var0}+${var1}
```

数组型常用基本操作

- 定义

```
var=(one two three four) #直接定义赋值
var=([0]="one" [2]="two") #定义，不连续赋值
var=(`cat dat.txt`) #从文件中读取数组赋值定义
```

- 取值，长度，删除，连接

```
echo ${var[0]} #输出数组var中的第一个值

echo ${var[@]} #输出数组var中的所有值
echo ${var[*]} #输出数组var中的所有值

echo ${#var[@]} #输出数组var长度
echo ${#var[*]} #输出数组var长度

unset ${var[0]} #删除数组var的第一个值
unset ${var}    #删除整个数组

#连接
var0=(hi hello)
var1=(one two three)
var=(${var0[@]} ${var1[@]})
```

if 常用条件判断

- if 的基本语法

```
if [ condition ]; then #注意[]内部空格
  语句
elif [ condition ]; then #注意[]内部空格
  语句
else
  语句
fi
```

- 文件目录判断

```
[ -d DIR ]  #如果 FILE 存在且是一个目录则为真
[ -e FILE ] #如果 FILE 存在则为真
[ -f FILE ] #如果 FILE 存在且是一个普通文件则为真
[ -r FILE ] #如果 FILE 存在且是可读的则为真
[ -s FILE ] #如果 FILE 存在且大小不为0则为真
```

- 字符串判断

```
[ -z STRING ] #如果STRING的长度为零则为真 , 即判断是否为空, 空即是真;
[ -n STRING ] #如果STRING的长度非零则为真 , 即判断是否为非空, 非空即是真;
[ STRING1 == STRING2 ] #如果两个字符串相同则为真 ;
[ STRING1 != STRING2 ] #如果字符串不相同则为真 ;
```

if 常用条件判断

- 数值判断

```
INT1 -eq INT2 #INT1和INT2两数相等为真 ,=
INT1 -ne INT2 #INT1和INT2两数不等为真 ,<>
INT1 -gt INT2 #INT1大于INT1为真 ,>
INT1 -ge INT2 #INT1大于等于INT2为真,>=
INT1 -lt INT2 #INT1小于INT2为真 ,<</div>
INT1 -le INT2 #INT1小于等于INT2为真,<=
```

- 复杂逻辑判断

```
-a #与
-o #或
! #非
```

- 判断实例 (猜字)

```
read -p "请输入0-10数字: " value #从外部读入数字
nvalue=‘echo ${value} | sed ’s/[0-9]//g’‘ #将数字都替换为空，赋值给nvalue
if [ ! -z ${nvalue} ]; then #判断nvalue是否为空，不为空，则不是数字
  echo "你输入的不是数字"
  exit 1
fi
if [ ${value} -eq $(($RANDOM%11)) ]; then #判断输入的数字和产生的随机数是否相等
  echo "猜对了"
else
  echo "猜错了"
fi
```

for 循环语句

- 数字性 for 循环 (循环输出 1 到 10)

```
#方法1
for((i=1;i<=10;i++)); do
  echo ${i}
done

#方法2
for i in {1..10}; do
  echo ${i}
done
```

- 字符性 for 循环

```
#案例
list="Hello World"
for i in ${list}; do
  echo ${i}
done
```

- 路径文件 for 循环

```
#案例
for i in `ls`; do
  echo ${i}
done
```

while 循环语句

- 基本语法

```
while condition; do
  语句
done
```

- 案例

```
#案例1
i=0; sum=0
while (( i < 10 )); do
  let sum+=i
  let ++i
done
echo "${sum}"

#案例2
cat FILE | while read line; do
  echo "${line}"
done

#案例3
i=0; sum=0
while [ ${i} < 10 ]; do
  sum=‘echo ${sum}+${i}|bc‘
  let ++i
done
echo "${sum}"
```

until 循环语句

- until 和 while 循环相反，当判断不成立的时候才进行循环。语法与 while 类似
- 基本语法

```
until condition; do
  语句
done
```

- 案例：

```
#案例1
i=0; sum=0
until (( sum > 50 )); do
  ((sum += i))
  ((i++))
done
echo "${sum}"

#案例2
i=0; sum=0
until [ ${i} > 10 ]; do
  ((sum += i))
  ((i++))
done
echo "${sum}"
```

break, continue 基本使用

- break 终止当前循环，continue 运行到当前继续下一轮循环
- 终止第 n 层循环

```
break n

#案例
for i in {1..5}; do
  for j in {6..10}; do
    echo "${i} ${j}"
    break 2
  done
done
```

- 运行到当前行后，继续第 n 层循环

```
continue n

#案例
for i in {1..5}; do
  for j in {6..10}; do
    echo "${i} ${j}"
    continue 2
  done
done
```

参数传递

- 从外部向脚本内部传递参数，可以在运行脚本命令行后面加上参数。脚本内部获取使用 $n (0 是文件名，1 则是第一个参数，2 则是第二个...)
- 特殊字符

```
$# #传递到脚本的参数个数
$* #以一个单字符串显示所有向脚本传递的参数
$$ #脚本运行的当前进程ID号
$! #后台运行的最后一个进程的ID号
$@ #与$*相同，但是使用时加引号，并在引号中返回每个参数
$- #显示Shell使用的当前选项，与set命令功能相同
$? #显示最后命令的退出状态。0表示没有错误，其他任何值表明有错误
```

- $@ 与 $* 的区别

```
#脚本test.sh, 运行bash test.sh 1 2 3 4
#测试$*
for i in "$*"; do
  echo $i
done

#测试$@
for i in "$@"; do
  echo $i
done
```

- 也可以用 read 从外部向内部传参数

函数

- 定义不带任何参数

```
#方法1
function func() {
  echo "BESIII"
}

#方法2
func() {
  echo "BEPCII"
}
```

- 参数返回，可以显示加：return 返回，如果不加，将以最后一条命令运行结果，作为返回值

```
function func() {
  read -p "输入第一个数字" a
  read -p "输入第二个数字" b
  return $(($a+$b))
}
```

- 函数中使用参数

```
function func() {
  a=$1 #第一个参数
  b=$2 #第二个参数
  return $(($a+$b))
}
#调用 func 1 2
```

输入输出重定向

- 重定向符号

```
command > file #将输出重定向到 file
command < file #将输入重定向到 file
command >> file #将输出以追加的方式重定向到 file
n > file #将文件描述符为 n 的文件重定向到 file
n >> file #将文件描述符为 n 的文件以追加的方式重定向到 file
n >& m #将输出文件 m 和 n 合并
n <& m #将输入文件 m 和 n 合并
```

- 案例

```
ls > file
ls >> file
cat file > newfile
```

Shell 小脚本实践

- 猜字小游戏
  要求：脚本随机给出一个数字 A。外部输入数字与 A 对比，并给出提示输入的数字比 A 大或者小，依次循环，直到猜对为止
- 文件操作
  要求：将文件 shelltest.dat 中的数字提取并相加输出

```
1 #文件shelltest.dat
2 A: 10000
3 B: 7000
4 C: 5300
5 D: 4320
```

Python 基础

Hello World

- 查看 Python 版本 (python3 不考虑向下兼容)

```
python -V
```

- 使用 vi 终端建立文件 helloworld.py

```
#!/usr/bin/env python3 #头声明解释器, 指定解释器的位置
print("Hello World") #屏幕输出"Hello World"
```

- 运行

```
python3 helloworld.py
```

Python 的基础语法

- 编码方式，默认情况都是 UTF-8 编码，字符串是 unicode 字符串，即

```
# -*- coding: UTF-8 -*-
```

- 标识符
  第一个字符必须是字母表中字母或下划线
  其他的部分有字母、数字和下划线组成
  大小写敏感
- 保留字

```
import keyword
keyword.kwlist
```

- 注释和 shell 一样，使用 # (注释还可以用三个双引号或者三个单引号)
- 使用缩进 (tab 键) 来表示代码块
- 数据类型：整数、长整数、浮点数和复数，字符串

基本数据类型

- 数字 (numbers): int、float、bool、complex

```
a = 2 #赋值
c, b = 2, 3 #同时赋值
type(a) #查看a的类型
#数字运算与C相似
```

- 字符串 (string)

```
#字符串赋值的方式
a = "hello world \t"
b = 'hello world \n'
c = """hello world"""

print(a, type(a), len(a)) #输出a, a的类型, a的长度, 发出转译
print(r"hello world \t") #不发生转译

print("hello"+" world") #输出"hello world"
print("hello"*2) #输出"hellohello"

print(a[0:2]) #输出"hel"
print(c[-3:-1]) #输出rld
```

基本数据类型

- 列表 (list)

```
a = ["hello", 2, 3, "world"] #列表内类型可以不同
b, d = [4, 5], [i for i in range(2, 10)]
c = a + b #支持列表直接相加 ["hello", 2, 3, "world", 4, 5]

print(a, len(a), a[1:3]) #输出 a, a的长度, a的第2到第4个元素

a[0] = "hi" #修改a的第一个元素
a[1:3] = [] #删除a的第2到第4个元素

#list内置其他操作, 例如append()、pop()...
```

- 元组 (tuple)

```
#元组元素不能修改, 除此之外与list相似
a = ("hello", 2, 3, "world")
a[0] = "hi" #非法的

c0 = () #空元组
c1 = ("hi", ) #一个元素, 需要在元素后添加逗号
#支持切片和相加 (和list类似)
```

基本数据类型

- 集合 (sets)

```
#无序不重复的集
a = {2, 3, 5, 3, 4} # 实际为 a = {2, 3, 5, 4}
b = set() #创建空的集合，不可以用b = {}
c = set("hello") # 等价于 c = {"h", "e", "l", "o"}
################ 支持运算 ################
a = set("hello")
b = set("hi")
a - b # a和b的差集 {"e", "l", "o"}
a | b #a和b的并集 {"h", "e", "l", "o", "i"}
a & b #a和b的交集 {"h"}
a ^ b #a和b中的不同时存在的元素 {""e", "l", "o"", "i"}
#max(tuple), min(tuple)可以直接寻找最大最小值
```

- 字典 (dictionaries)

```
#映射类型，无序的 键:值 集合 关键字必须为不可变类型，同字典中关键字不同
a = {} #创立空字典
b = {"A": 2, "B": 3, "C": 4} #创立非空字典
print(b["A"]) #输出字典b中的关键字 “A” 对应的值
del b["A"] #删除
b["D"] = 7 #添加
b.keys(), b.values() #b字典中的所有关键字,值
########## 字典其他定义方式 ##############
a, b = dict(A=2, B=3, C=4), dict([("A", 2), ("C", 4), ("B", 3)])
c = {x: pow(x, 2) for x in (1, 2, 3)} #等价于 {1: 1, 2: 4, 3: 9}
#字典有内置函数clear(), update(dict), copy(), get(key, default=None)...
```

条件控制

- 语法

```
if condition1:
  语句
elif condition2:
  语句
else:
  语句

#案例
n = 10
if n < 10:
  print("n < 10")
elif n == 10:
  print("n == 10")
else:
  print("n > 10")

#常用的操作运算符
< #小于
<= #小于或等于
> #大于
>= #大于或等于
== #等于，比较对象是否相等
!= #不等于
```

循环语句

- while 循环

```
while condition:
  语句

#案例
n, sum = 0, 0
while n < 100:
  sum += n
  n += 1
print(sum)
```

- for 循环

```
for varible in sequence
  语句
else:
  语句

#案例
for i in range(0, 10):
  print(i)
```

continue, break, pass

- continue 语句被用来告诉 Python 跳过当前循环块中的剩余语句，然后继续进行下一轮循环
- break 语句可以跳出 for 和 while 的循环体。如果你从 for 或 while 循环中终止，任何对应的循环 else 块将不执行
- pass 语句什么都不做。它只在语法上需要一条语句但程序不需要任何操作时使用

```
1  #案例
2  for i in range(2, 10):
3    for j in range(20, 200, 5):
4      pass
5      print("pass: %i, %i"%(i, j))
6      if j == 35:
7        continue
8        print("continue: %i, %i"%(i, j))
9      if i == 5:
10       break
11       print("break: %i, %i"%(i, j))
```

函数

- 函数定义

```
def 函数名(参数):
  函数体

#案例
def myfunc(resonance, mass, width):
  print(resonance, mass, width) #输出共振态，质量，宽度
  return resonance, mass, width #返回共振态，质量，宽度

def func(*args): #可变个数的参数列表
  for i in args:
    print(i)

#调用
res, m, sigma = myfunc("J/psi", 3.097, 0.0000929)
res, m, sigma = myfunc(resonance = "J/psi", mass = 3.097, width = 0.0000929)
func(2, 3, 4, 5, 6)
func(4, 5, 6)

#**kwargs 参数形式
def hello(**kwargs):
  key = kwargs.keys()
  value = kwargs.values()
  print(key)
  print(value)
#调用 hello(a = 3, b = 4, c = 6)
```

模块

- 模块是一个包含所有你定义的函数和变量的文件，其后缀名是.py。模块可以被别的程序引入，以使用该模块中的函数等功能

```
1  import random
2  print(random.randint(1, 10)) #输出1到10的一个整数随机数
3
4  import random as rd
5  a = rd.randint(1, 10)
6
7  from random import randint
8  a = randint(1, 10)
```

类

- 类（class）：用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例
- 类变量：类变量在整个实例化的对象中是公用的。类变量定义在类中且在函数体之外。类变量通常不作为实例变量使用
- 数据成员：类变量或者实例变量用于处理类及其实例对象的相关的数据
- 方法重写：如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（override），也称为方法的重写
- 局部变量：定义在方法中的变量，只作用于当前实例的类
- 实例变量：在类的声明中，属性是用变量来表示的。这种变量就称为实例变量，是在类声明的内部但是在类的其他成员方法之外声明的
- 继承：即一个派生类（derived class）继承基类（base class）的字段和方法。继承也允许把一个派生类的对象作为一个基类对象对待
- 实例化：创建一个类的实例，类的具体对象
- 方法：类中定义的函数
- 对象：通过类定义的数据结构实例。对象包括两个数据成员（类变量和实例变量）和方法

```python
class MotherParticle(): #定义类
  ''' 母粒子的类 '''
  _name, _mass, _width = "", -1.0, -1.0 #定义类成员

  def __init__(self, name, mass, width): # 类构造
    self._name, self._mass, self._width = name, mass, width

  def print(self): #类成员函数, 输出
    print("Particle name: %s mass: %.5f, width: %.5f"%(self._name, self._mass, self._width))

  def print_particle(self): #类成员函数, 输出
    print("Mother name: %s mass: %.5f, width: %.5f"%(self._name, self._mass, self._width))

class ChildParticle(MotherParticle):
  ''' 子粒子的类 '''
  _name, _mass, _width = "", -1.0, -1.0 #定义类成员

  def __init__(self, name, mass, width): # 类构造
    super().__init__(name, mass, width) # 初始化父类 MotherParticle
    self._name, self._mass, self._width = name, mass, width

  def print_particle(self): #类成员函数, 输出
    print("Child name: %s mass: %.5f, width: %.5f"%(self._name, self._mass, self._width))

# 测试
a = ChildParticle("Jpsi", 3.097, 0.0000929)
a.print()
a.print_particle()
```

异常测试

- try ... except ...

```
#案例
try:
  x = int(raw_input("enter a number: "))
  print(x)
except ValueError: #异常抛出语句，也可直接使用except
  print("this is not a valid number"):

#异常类可以自己定义
class MyError(Exception):
  def __init__(self, value):
    self.value = value
  def __str__(self):
    return repr(self.value)
#测试
try:
  raise MyError(1)
except MyError as err:
  print("my test error value: ", err.value)
```

异常测试

- else, finally

```
1  #如果一个异常在 try 子句里（或者在 except 和 else 子句里）被抛出，而又没有任何的 except 把它截
       住，那么这个异常会在 finally 子句执行后再次被抛出
2  try:
3    result = x / y
4  except ZeroDivisionError:
5    print("division by zero!")
6  else:
7    print("result is", result)
8  finally: #一定会执行
9    print("executing finally clause")
```

file 常用操作

- 读写

```
案例:
f = open("log", "w") #mode 可以是 'r' 如果文件只读, 'w' 只用于写 (如果存在同名文件则将被删除)
      , 和 'a' 用于追加文件内容; 所写的任何数据都会被自动增加到末尾. 'r+' 同时用于读写。 mode
      参数是可选的; 'r' 将是默认值
f.write("hehehe") #写入文件
f.read() # 文件的所有内容都将被读取并且返回
f.readline() #文件中读取单独的一行
f.readlines() #文件中包含的所有行
f.close #关闭文件并释放系统的资源
```

- 推荐结合 with ... as ...

```
with open("log") as f:
  f.write("hhhh")
```

- 结合 for

```
with open("log") as f:
  for i in f:
    print(i)
```

python 中使用 shell (subprocess)

- subprocess 包中定义有数个创建子进程的函数，这些函数分别以不同的方式创建子进程
- subprocess 还提供了一些管理标准流 (standard stream) 和管道 (pipe) 的工具，从而在进程间使用文本通信

```
1  #案例
2  import subprocess
3  def cmd(command):
4    subp = subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stderr=subprocess.PIPE
       ,encoding="utf-8")
5    subp.wait(2) #等待子进程终止
6    if subp.poll() == 0: #进程状态判断
7      print(subp.communicate()[1])
8    else:
9      print("失败")
```

PyROOT 小案例

```
#创建ROOT文件并写入数据
#!/usr/bin/env python
#-*- coding: UTF-8 -*-

import ROOT as root
from array import array

rootfile = root.TFile("test.root", "recreate")
tree = root.TTree("tree", "test tree")

br1, br2 = array("d", [-999.]), array("d", [-999.])
tree.Branch("br1", br1, "br1/D")
tree.Branch("br2", br2, "br2/D")

rand = root.TRandom()

i = 0
while i < 1000:
  i += 1
  br1[0] = rand.BreitWigner(10.0, 1.0)
  br2[0] = rand.Gaus(4.0, 1.0)
  tree.Fill()
tree.Write()
rootfile.Close()
```

小程序编写实践

- 猜字小游戏
- 读取 ROOT 文件画图