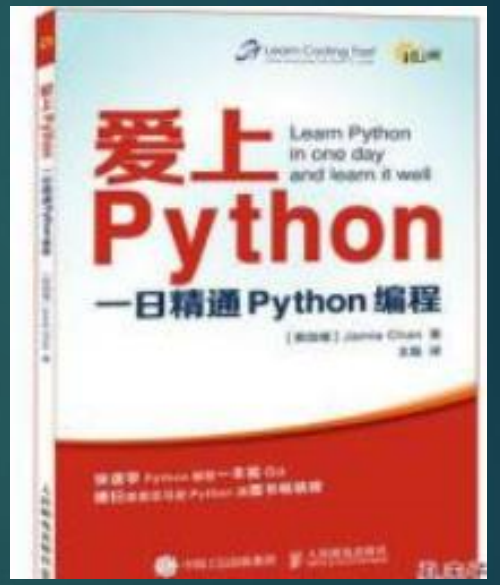
<PKU 통계스터디 첫 번째 수업>

파이썬 기본 어법배우기



참고문헌: 爱上 python（人民邮电出版社）

2018. 03. 03 토요일

발표자 – 이수민 이준호

주석기능-기록이 명령에 영향을 미치지 않는법

1. (한국어 입력을 하려면 "# -- coding: UTF-8 --" 이것이 첫번째 항에 들어가야함)

#이것은 주석이다

#주석은 샵으로

#샵은 프린트되지 않는다

예시) x=3 #x값은 열매반 학우들의 머릿수이다

2.

'''

이것은 주석이다

주석은 작은 따옴표 세개

작은 따옴표 세개사이는 프린트되지 않는다

'''

변수 정의하는 법

- 새로운 변수를 만들 때마다, 새 값을 넣어주기(값에 관하여..)

- 예시1)

userAge, username = 30, ‘Peter’

- 예시2)

userAge = 30

username= ‘Peter’

예시1과 예시2는 같은 내용

부여값 부호 赋值符号

- userAge=0 이 뜻은 userAge와 0이 같다라는 뜻이 아니라, 0이라는 값을 부여한 것

- 순서에따라, x=y와 y=x의 의미는 다르다

예)
X=5
Y=10
X=Y

Print (X,Y)

#이것의 정답은?

기본 조작부호 基础操作符

- **x=5, y=2 라고 가정합니다.**

- 덧셈: x+y=7
- 뺄셈: x-y=3
- 곱하기: x*y=10
- 나누기: x/y=2.5
- 整除：x//y=2（정수만 보류하는 나누기）
- 나머지：x%y=1（当5除以2时得到的）
- 승의 계산：x**y=25

+=, -=, *=

- **+=, -=, *=**
  - "x~=2"와 "x= x~2"는 같은 의미를 가집니다.
- 예시1)

x=x+2

x+=2

- 예시2)

x=x-2

x-=2

파이썬에서의 변량(변수)타입

- int (integral의 약자)：소수점이 없는 정수(ex. -5, -4, 0, 5, 7 등)
- float ：소수점이 있는 수(ex. 3.0, 1.234, -0.023, 12.01)
- string：글자 형식의 변량입니다 (ex. “hi”, ‘hi’, “this is a apple”, “30.12”)

- 예시)

userAge=‘30’ 이라 쓴다면, userAge는 string입니다.
반대로, 만약 userAge=30이라고 쓴다면(따옴표 없이), 이것은 int입니다.
+ 기호를 사용하여, 스트링을 연결 할 수 있습니다.

- 예시) “Peter”+”Lee”은 “PeterLee”

Python의 변수타입전환(类型转换)

- int(),float(),str()함수는 서로 전환이 가능

- int()함수는 다른 두 유형의 변수를 int로 변경합니다.

예) int(5.71283) #결과는 뭘까요?

string을 int으로 변형하기 위해, int("4")를 4로 변경 가능.

- ✓ 하지만 int("hello") or int("4.22321")과 같은 경우는 틀린 결과를 가져올 수 있음 #why?

- float()함수는 변수를 float 형(型)으로 바꿔줌.

예)float(2) 혹은 float("2 ")의 결과는 2.0 #float("2.09")출력값으로 나온 2.09의 종류는?

- Str()함수는 변수의 형식을 스트링으로 바꿔줍니다.

예)str(2.1)의 결과는 "2.1"

string의 내재함수 두개 : upper(), lower()

- 함수(function)란?
  - 어떤 값을 input 했을때, 어떤 값을 output하는 모듈.
    - 수학의 실제 함수와는 달라서, 아무런 값을 받지 않아도 되게 설정 가능하며, 아무런 일도 하지 않는 함수도 설정할수 있고, 아무런 값을 output하지 않는 함수를 설정할 수도 있습니다.
- 내재함수란?
  - 어떤 python의 자체적으로 쓰여진 class 모듈의 함수.
  - 이는 사용자가 직접 정의한 함수와 '저자'가 다르다는 점에서 내재함수이냐, 아니냐가 정의 내려집니다.
- String.upper(), String.lower()
  - str = "BeiJINg"
  - Str1 = str.upper()

변수의 종류가 중요한 이유

- 다른 변수끼리의 연산이 불가능 하기 때문입니다
  - 파이썬은 C++보다는 유동적인(flexible)한 컴퓨터 언어이기 때문에, float와 int의 연산이 가능할지 모르나, 반환하는 값의 정확도는 보장할수 없습니다
  - 또, 많은 경우에 에러를 동반하기도 합니다. (예를 들어 float와 string의 연산이 불가능 합니다. string끼리의 나눗셈, 곱셈역시 불가능 합니다)
  - 따라서, 하나의 코드를 실행시켰을때, type관련 에러가 뜬다면, 해당코드에서 정의한 변량의 유형에 대해 살펴볼 필요가 있습니다.
  - 본인이 정의한 변량을 type() 라는 파이썬에 내재 되어 있는 함수로, 해당 변량의 종류를 찾아볼수 있습니다 (예, a= 3.0, print(type(a))

list 변량

- **userAge=[21,22,23,24,25]**

- userAge[index] 형식으로 list안의 원소들을 가져올수 있습니다.
- userAge[0]=21, userAge[1]=22
- userAge[-1]=25, userAge[-2]=24
- userAge[2:4]= 23,24
- 새로운 값 설정하기: userAge[1]=5라고 하면, userAge=[21,5,23,24,25]
- 추가하기: userAge.append(99)라고 하면, userAge=[21,22,23,24,25,99]
- 삭제하기: del userAge[2]라고 하면, userAge=[21,22,24,25]

- 결론 : list형식의 변량은, 쓰기가능한(writeable) 변수 입니다. 이는 곧 배우게 될 tuple형식의 변량과 가장 주요한 차이점 입니다.
- 문제 : 어제 숙제(평균값 구하기) list와, for문을 써서 풀기.

tuple

- Tuple과 list는 많은 점에서 같으나, index의 값을 고칠수 있냐, 없냐의 주요한 차이점이 있습니다.

- Tuple변량은 “()” 로 정의 합니다. (list의 경우 “[]”로 정의 했습니다.)

- index로 변량의 값을 반환하는 법은 list와 동일 합니다.

- 예 :

```
test = (‘Jan’, “Feb”, “Mar”, 4, ‘Apr’)
print(test[0])   #?
print(test[-1])   #?
```

dictionary 字典

- dictionary형식의 변량은 “{}”로 정의 합니다. 그리고 이것의 표준적인 정의 형식은 :

dictionaryName = {key1: data1, key2: data2, …} (key 는 “string”형식이어야 합니다)

- userNameAndAge={“Peter”:38, “John”=51, “Alex”=13, “Alvin” = “not available”}
- print(userNameAndAge[“John”]) 의 결과는 51
- #userNameAndAge[“John”]=21 (이런 형식으로 특정 key에 대응하는 data값을 변경할수 있습니다)
- #userNameAndAge[“Tom”]=40 (새로운 key와 data 추가)
- #del userNameAndAge[“Alex”] (특정 key지우기)

정리 : list, tuple, dictionary

- 우선 이들은 여러가지 변량들을 하나의 변량안에 담을수 있다는 점에서 유사합니다.
- 하지만, 이들을 통해 데이터를 처리 하는 방법이 다르기에, 서로 구분됩니다.

- 공통적으로 가능한 조작/특성:

print() 함수로 모든 내용 print하기

Name[index]로 해당 index내용 값 찾기.

저장할수 있는 원소 종류의 제한이 없습니다.

User interactive coding : input 함수

- 이는, 완성된 코드를 실행 시킬때, 실행자가 코드 실행실행 과정에 개입할수 있게 합니다. 예를 들어, 어떤 특정한 변수에 원하는 값을 부여할수 있습니다.

Example :

A = “Hello, glad to meet you all!”

MyName = input(“please input your name : ”)

MyAge = input(“input your age, please : ”)

print(A, “My name is”, MyName)

print(A, “My name is”, MyName, “I am”, MyAge, “years old.”)

print(A,”My name is %s and I am %s years old.” %(MyName, MyAge))

print(A, “My name is {} and I am {} years old.” .format(MyName, MyAge))

등호/부등호 기법 : ==, !=, >, <, >=, <=

- 컴퓨터는 비트로 이루어져 있습니다, 비트는 0,1.
- 파이썬에서의 등호/부등호 기법 : (1) 같다 “==“, (2) 다르다 “!=” (3) “>”, (4) “<“, (5) “>=”, (6)”<=”

- False = 0, True =1 이라고 보시면 되겠습니다(대/소문자 주의!), 아래 예 :
- A = (5==4)
- B = (4==4)
- print(A,B)
- print(A==B)
- print(A!=B)
- print(A>B)
- print(A<B)
- print(A<=B)

논리 연산자 : &, |

- “&”는 ~~와 라는 의미이고, “|”는 ~~혹은 이라는 의미 입니다.
- A = (3==2)
- B = (3 != 1)
- print(A & B)
- Print(A | B)

- 의미 생각해 보기.
- 논리 연산자들은 등호/부등호와 자주 같이 사용됩니다. (for문에서 확인하시길 바랍니다)

조건구 : if, elif, else (줄맞추기!!)

Only executed once if condition is satisfied(True), and jump out of "if"

if & else combination

```
if True:
    print("True")
else:
    print("this is not printing")
```

==========

```
if False:
    print("False")
else:
    print("this is not printing")
```

Generalization

```
if condition:
    do ~~~~~
else:
    do ~~~~
```

==========

This is usally used for the case, either the event is true or not, is covering all of the events reasonably.

if & elif & else combination (generalized)

```
if condition 1:
    do ~~~~
elif condition 2:
    do ~~~~
elif condition 3:
    do ~~~~
.
.
.
else :
    do ~~~~
```

This is used for the case, separating several events into each reasonable condition, and "do" if specific condition is satisfied.

Examples for “if”

- General “if”

(try to focus on True/False on each contidion)

```
A ,B = 10, 20
if A>B:
    print(“A is bigger than B”)
elif A<B:
    print(“A is smaller than B”)
else:
    print(“A and B are even”)
```

- Interfaced “if”

```
Me = 10
Num1 = 12 if Me>=12 else Me
print(Num1)
Num2 = 10 if Me!=10 else 0
print(Num2)
```

The “for” loop

range(10) contains numbers from 0 to 9

Typical “for”

```
K=0
for j in range(10):
    L = j + 1
    K = K + L

print(K)
```

“list” interfaced “for”

(try to focus on True/False)

```
A = [10,20,30,40]
K = 10
for K in A:
    print( K,“ is contained”)
    k += 10
print(K)
```

“continue” & “break” involved “for”

```
KL = 0
for KL in range(11):
    print(KL)     # what if this locates under “elif”
    if KL==4:
        continue
    elif KL>=8:
        break
    else:
        pass
print(KL)
```

The “range()” function

- range() function makes “list”
- range(5) → [0,1,2,3,4]
- range(3,10) → [3,4,5,6,7,8,9]
- range(4,10,2) → [4,6,8]

The “while” loop

General case

(1)

```
while True:
    do ~~~~
```

(2)

```
While True:
    do ~~~~
    if(~):
        break
```

==========

The “while” executes sub-command of its, until (1)the condition of “while” is no longer “True” , (2) or meets “break”

Typical example 1

```
Counter = 5

while Counter > 0 :
    print(“counter = “,Counter)
    Counter = Counter - 1
```

Typical example 2

```
Counter2 = 5

while Counter2>0:
    print(“counter2 = “, Counter2)
    Counter2 = Counter2 + 1
    if( Counter2 >= 15):
        break
```

==========

Image what happens if there is no “break”.

“try” & “except”

General Case

```
try:
    do ~~~~
except:
    do ~~~
```

Typical example1

```
try:
    answer = 12/0
    print(answer)
except:
    print(“error!”)
```

==========

Image result of the code.