Ch 3 파이썬 환경

목차

- 파이썬 소개
- 변수와 표현식
- 자료형
- 함수
- 클래스와 객체
- 입출력과 예외
- 모듈

3.1 파이썬 소개

- **Python 은 1991년에 귀도 반 로섬 (Guido Van Rossum)** 이 만든 프로그래밍 언어
- 특징
  - **C, Java** 등과 같은 고급 언어이며 인터프리터 언어
    - 프로그램 코드를 작성하면서 실행 가능하며 변수 타입을 설정 불필요
  - 문법이 매우 쉽고 직관적→배우고 이해하기 쉬우며 코드가 간결
  - 객체 지향적인 클래스를 지원
  - 특정한 기능을 모듈로 패키지화 및 다른 언어 모듈 포함 확장 가능
  - 현재 **3.3** 버전 제공
    - 호환성 문제로 **2.7** 버전도 함께 제공
  - www.python.org

파이썬 소개

- 라즈베리파이
  - 파이썬을 위해 **IDLE** 및 **IDLE3** 제공
- **IDLE**
  - 파이썬을 위한 통합개발환경 **(IDE)**
  - 다중 윈도우 , **syntax highlighting** (문장형식 색/강조 표시), 자동완성 및 들여쓰기 지원
  - 문장형식 색/강조 표시 되는 파이썬 쉘 기능
  - 순차실행, **breakpoint**, 스택보기기능을 포함한 통합 디버깅
  - 파이썬 **3.x** 버전을 위한 **IDLE3** 프로그램 따로 제공
  - 단점
    - 클립보드 복사 기능 없음
    - 라인 번호 표시 안됨

3.2 파이썬 실행

▣ 파이썬 쉘 또는 IDLE 실행

```
$ python
…
>>> print (“Hello World”)
Hello World
>>> quit()
$
```

◈ 파이썬 3 에서는 **print(“Hello World”)** 를 입력해야 함

▣ 표현식 및 수학식 입력 가능

```
>>> 1 + 2 + 3
6
>>> (0 < 1) and (2 < 3)
1
>>> not (1 == 2)
1
```

파이썬 실행

▣ 스크립트 실행 가능

◈ 스크립트 파일(helloworld.py) 작성

```
#!/usr/bin/env python
print ("Hello World")
```

◈ 실행

```
$ python helloworld.py
Hello World
$ ./helloworld.py
Hello World
$
```

3.3 변수와 표현식

- 변수에는 정수, 실수, 문자열 등 다양한 자료형을 대입하거나 자료형 변경 가능

```
>>> n = 1
>>> name = 'Kim'
>>> n = n + 2
>>> value = 1.2 * n
>>> print ("%f is 1.2 * %d" % (value, n))
3.600000 is 1.2 * 3
>>>
```

◈ 예제 – 피보나치 수열 출력

```
#!/usr/bin/env python
num = 1
prev = 0
cur = 1
while num < 10:
    next = cur + prev
    print ("%2d %d" % (num, next))
    prev = cur
    cur = next
    num += 1
```

```
$ python pibonacci.py
1 1
2 2
3 3
4 5
5 8
6 13
7 21
8 34
9 55
```

변수와 표현식

변수

- 임의의 식별자 사용 가능
- 다음 예약어는 사용 불가능

and	as	assert	break	class	continue	def	del
elif	else	except	exec	finally	for	from	global
if	import	in	is	lambda	nonlocal	not	or
pass	print	raise	return	try	while	with	yield

자료형

▣ 상수

◈ 숫자상수

- **int, long** – 정수
- **float** - 부동소수점 수, 실수
- **complex** – 복소수
- **boolean** - 부울상수 **(True, False)**

◈ 문자열 상수 - ‘ ‘, “ “, ‘’’ ‘’’ 사용/ 인덱스연산자 [] 사용

```
>>> a = “Hello World”
>>> a[0]
‘H’
>>> a[5:8]
‘ Wo’
>>> a[:2]
‘He’
>>> a[-1]
‘d’
>>> a * 2
‘Hello WorldHello World’
>>> b = “ Goodbye”
>>> a + b
‘Hello World Goodbye’
```

자료형

▣ 리스트 (list)

- ◈ 임의의 객체들을 순서대로 저장한 순서열
- ◈ [ ] 대괄호 사용
- ◈ 항목 추가 – **append()** 함수, 항목 삽입 – **insert()** 함수 사용
- ◈ 리스트연결 - + 연산자
- ◈ 빈리스트 생성 – **[]** 또는 **list()** 함수 사용

```
>>> numbers = [0, 1, 2, 3]
>>> names = [“Kim”, “Lee”, “Park”, “Choi”]
>>> numbers[0]
0
>>> names[2:]
[“Park”, “Choi”]
>>> numbers[-1]
3
>>> numbers + names
[0, 1, 2, 3, 'Kim', 'Lee', 'Park', 'Choi']
>>> empty = []
>>> empty
[]
```

자료형

▣ 튜플 (tuple)

- ◈ 리스트와 비슷하지만 리스트와는 달리 생성된 항목 변경 불가
- ◈ () 괄호 사용
- ◈ 항목추출 - 리스트처럼 숫자인덱스 사용하거나 콤마(,) 사용

```
>>> person = ( “Kim”, 24, ‘male’)
>>> a = ( )
>>> a
()
>>> b = (person, )
>>> b
(('Kim', 44, 'male'),)
>>> name, age, gender = person
>>> name
'Kim'
```

자료형

▣ 집합 (set)

- ◈ 객체들을 순서없이 모은 것
- ◈ **set()** 함수로 생성
- ◈ 숫자 인덱스로 추출 불가 및 요소 중복 불가
- ◈ 합집합, 교집합, 차집합 , **add(), update(), remove()** 연산 가능

```
>>> even = set([0,2,4,6,8])
>>> hello = set("Hello")
>>> hello
set(['H', 'e', 'l', 'o'])
>>> s = even | hello
>>> p = even & hello
>>> even.add(10)
>>> hello.remove('e')
```

자료형

▣ 사전 (dictionary)

- ◈ 키 (key) 로 색인되는 객체들을 저장한 해시테이블
- ◈ {key:value} 와 같은 형식 사용
- ◈ 키 인덱스 연산자로 각 요소에 접근
- ◈ 빈 사전 생성 – {} 또는 dict() 함수 사용

```
>>> me = { "name" : "Kim", "age" : 44, "gender" : "male" }
>>> myname = me["name"]
>>> me["age"] = 35
>>> dict = {}
```

자료형

- 파이썬의 모든 데이터는 객체 모델 사용
- 객체의 속성
  - 식별(identity)
    - 객체가 생성된 후 다른 객체와 구별되는 식별자
    - “is” 연산자로 구별 가능
    - id() 함수로 획득 가능
  - 타입(type)
    - 객체가 어떤 값을 가지며 어떤 연산이 적용될 수 있는지 구분
    - type() 함수로 회득 가능

```
>>> type(10)
<type 'int'>
>>> type([1,2])
<type 'list'>
>>> type(type(10))
<type 'type'>
```

  - 값(value)

자료형

▣ 기타 자료형

◈ None

- ➢ **Null** 객체
- ➢ 아무 값이나 속성을 가지지 않음
- ➢ **boolean** 표현식의 **False** 값으로 평가
- ➢ 주로 값을 반환하지 않는 함수나 생략 가능한 인자의 기본 값

◈ 호출가능한(Callable) 객체

- ➢ 함수, 클래스, 메소드

구문

들여쓰기

- 파이썬에서는 엄격한 들여쓰기를 통해 조건문, 반복문, 함수, 클래스 등의 코드 블록을 구분
- 가독성이 뛰어남

```
# Setup
n = 0

# Loop
while True:
    n = n + 1
    # The % is the modulo operator
    if ((n % 2) == 0):
        print(n)
```

3.4 프로그램 제어

▣ 조건문

◈ if ~ else 구문

```
if 표현식:
    if_구문
else:
    else_구문
```

```
if a > b:
    print ("max is %d" % a)
else:
    print ("max is %d" % b)
```

◈ 다중 조건문 (if ~ elif ~ else 구문)

```
if 표현식1:
    if_구문
elif 표현식2:
    elif_구문1
else:
    else_구문
```

```
if i > 0:
    print ("this is positive")
elif i == 0:
    print ("this is zero")
else:
    print ("this is negative")
```

프로그램 제어

▣ 반복문

◈ while 구문

```
while 표현식:
    while_구문
```

➢ 예제 – 피보나치 수열 출력

```
#!/usr/bin/env python
num = 1
prev = 0
cur = 1
while num < 10:
   next = cur + prev
   print "%2d %d" % (num, next)
   prev = cur
   cur = next
   num += 1
```

```
$ python pibonacci.py
1 1
2 2
3 3
4 5
5 8
6 13
7 21
8 34
9 55
```

프로그램 제어

▣ 반복문

◈ for 구문

```
for 변수 in 목록:
    구문
```

- C언어의 **for** 구문이라기보다 **c#** 의 **foreach** 구문과 비슷
- 리스트, 튜플, 문자열과 같은 객체의 모든 원소들에 대해 반복 실행

```
for i in [0, 1, 2, 3, 4]:
  print ("%d ^ 2 = %d" % (i, i*i))
```

- 특정한 횟수만큼 실행하려면 **range()** 함수 사용

```
for i in range(5) :
    print ("%d ^ 2 = %d" % (i, i*i))
c = range(1,6) # c = 1,2,3,4,5
```

프로그램 제어

▣ 반복문

◈ break 구문

➢ 반복문을 빠져나옴

```
for i in [0, 1, 2, 3, 4]:
   if i % 2 != 0:
      break
   print ("%d ^ 2 = %d" % (i, i*i))
```

◈ Continue 구문

➢ 반복문의 처음으로 이동

```
for i in [0, 1, 2, 3, 4]:
   if i % 2 != 0:
      continue
   print ("%d ^ 2 = %d" % (i, i*i))
```

◈ Pass 구문

➢ 아무 실행도 하지 않음

```
for i in range(5) :
    pass
```

3.5 함수

▣ 함수

- ◈ **def** 문을 사용하여 정의
- ◈ 범위는 들여쓰기로 쓰여진 부분이며 지역 범위를 가짐
- ◈ 함수 호출은 함수 이름을 쓰고 인자들을 괄호로 둘러쌈

```
def min(a, b):
   if a > b:
     return b
  else:
     return a
c = min(10, 20)
```

- ◈ 여러 인자값 반환 가능 – 튜플 사용

```
def divide(a, b):
     return (a/b, a%b)
q, r = divide(3,2)
print q, r
1 1
```

함수

변수의 범위

- 지역 변수는 함수가 반환되면 메모리에서 삭제
- 함수 안에서 전역변수를 참조하거나 값을 수정하려면 **global** 키워드를 사용

```
m = 0
n = 1
def func():
    global n
    m += 1
    n += 1
func()
print (m, n)
```

0 2

함수

중첩 함수

- 함수 안에서 함수 정의 가능
- 중첩 함수에서 외부 지역변수 참조 가능

```
def counter(max):
    t = 0
    def output():
        print ("t = %d" % t)
    while t < max:
        output()
        t += 1
counter(10)
```

재귀함수

```
def factorial(n):
    if n <= 1:
        return 1
    else:
        return n * factorial(n - 1)
```

함수

lambda 함수

◈ 지역 변수는 함수가 반환되면 메모리에서 삭제

lambda 인자목록 : 반환표현식

```
a = lambda x, y : x+y
b = a(1,2)
```

함수

클로저(closure)

- 함수의 내용과 함께 함수가 저장된 내부 환경을 제공하는 것
- 함수의 실행을 늦추거나 내부 상태를 저장하는 데 유용하게 사용
- 함수 평가 예 (클로저 아님)

```
#func.py
def func(f):
    return f()
```

```
import func
def hello():
    return "Hello World"
print func.func(hello)
```

Hello World

- 클로저 예

```
def calc(a):
    def add(b):
        return a + b
    return add
sum = calc(1)
print sum(2)
```

3

```
def counter2():
    t = [0]
    def increment():
        t[0] += 1
        return t[0]
    return increment
timer = counter2()
timer()
```

1

```
timer()
```

2

함수

▣ 장식자(decorator)

- ◈ 다른 함수나 클래스를 포장하는 함수 (클로저로 구현)
- ◈ 포장된 함수나 클래스를 변경하여 실행 가능
- ◈ '@' 기호 사용
- ◈ 예

```
def deco(f):
    def func(x):
        print "Begin", f.__name__
        print f(x)
        print "End", f.__name__
    return func
```

```
@deco
def incr(x):
    return x + 1
```

=

```
def incr(x):
    return x + 1
incr = deco(incr)
```

<출력>

```
$ incr(2)
Begin incr
3
End incr
```

함수

▣ 장식자(decorator)

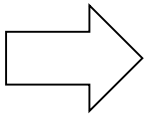
```
def helloBye(func):
    def wrapper():
        print "hello"
        func()
        print "bye"
    return wrapper

@helloBye
def f1():
    print "f1() runned."

def f2():
    print "f2() runned."

@helloBye
def f3():
    print "f3() runned."

if __name__ == '__main__':
 f1()
    print "------"
    f2()
    print "------"
    f3()
```



```
<출력>
hello
f1() runned.
bye
------
f2() runned.
------
hello
f3() runned.
bye
```

함수

생성기(generator)

- yield 키워드를 사용하여 만든 클로저
- next() 메소드를 호출하면 yield 까지만 실행
- 예

```
>>> def counter3(max):
...     t = 0
...     while t < max:
...         yield t
...         t += 1
...     return
>>> timer = counter3(5)
>>> timer.next()
0
>>> timer.next()
1
```

```
for i in counter3(5):
    print I

0
1
2
3
4
```

함수

코루틴(coroutine)

- yield 키워드를 입력값으로 사용
- send() 메소드를 호출하여 yield 문으로 입력
- 예

```
>>> def handler():
...     print "Initialize Handler"
...     while True:
...         value = (yield)
...         print "Received %s" % value
>>> listener = handler()
>>> listener.next()
Initialize handler
>>> listener.send(1)
Received 1
>>> listener.send("message")
Received message
```

함수

▣ 리스트 내포 (list comprehension)

```
>>> numbers = [0, 1, 2, 3, 4]
>>> evens = [2*i for i in numbers]
>>> print evens
[0, 2, 4, 6, 8]
>>> sum(evens)
20
```

▣ 생성기 표현식(generator expression)

◈ 리스트 내포와 비슷하지만 생성기이므로 메모리 상태 유지

```
>>> evens2 = (2*i for i in numbers)
>>> evens2
<generator object <genexpr> at 0xb715ec84>
>>> evens2.next()
0
>>> evens2.next()
2
>>> sum(evens2)
20
```

3.6 클래스와 객체

- 파이썬 프로그램의 모든 값은 내부 데이터와 메소드로 구성된 객체임
  - 객체 제공 메소드 확인 – **dir()** 메소드 사용

```
>>> numbers = [10, 20]
>>> dir(numbers)
['__add__', '__class__', ......
.....
 'appeld', 'count', ...
```

- 객체지향 프로그래밍 지원 - 클래스 **(class)**
  - **class** 키워드로 정의하고 들여쓰기로 쓰여진 부분
  - 메소드**(method)**, 변수**(class variable)**, 속성**(property)**으로 구성

클래스와 객체

▣ 클래스 정의

◈ 객체를 만들기 위한 틀

```
class Person(object):
    total = 0
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def getAge(self):
        return self.age
```

◈ 내부 멤버에는 '.' 로 접근

```
Person.total
Person.__init__
Person.getAge
```

클래스와 객체

클래스 인스턴스 생성

- 클래스 객체를 함수로서 호출하면 인스턴스 생성
- 이 때 __init__() 메소드에 인스턴스를 전달하고 초기화

```
john = Person("John Doe", 35)
John.name
John Doe
john.age
35
```

상속

- 상위 클래스로부터 하위 클래스를 상속할 수 있음
- 하위 클래스는 상위 클래스의 속성을 상속받으며 재정의 가능
- 다중 상속 가능

```
class Man(Person):
    gender = 'male'
...
class KoreanMan(Man, Korean):
...
```

클래스와 객체

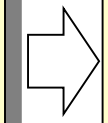
▣ 클래스 예제

- ◈ 클래스: 객체지향언어 기본 단위 (이름: **CamelCase** 사용)
- ◈ 클래스 변수: 공통 사용
- ◈ 메서드(**def**): 클래스의 행위
- ◈ 초기자(_ _**init**_ _): 새 객체 생성

```
class Rectangle:
    count = 0  # 클래스 변수

    # 초기자(initializer)
    def _ _init_ _ (self, width, height):
        # self.* : 인스턴스변수
        self.width = width
        self.height = height
        Rectangle.count += 1

    # 메서드
    def calcArea(self):
        area = self.width * self.height
        return area
```



```
# 인스턴스 생성
r = Rectangle(2, 3)

# 메서드 호출
area = r.calcArea()
print("area = ", area)

# 인스턴스 변수 엑세스
r.width = 10
print("width = ", r.width)

# 클래스 변수 엑세스
print(Rectangle.count)
print(r.count)
```

클래스와 객체

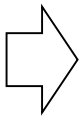
▣ 클래스 상속

- ◈ 파생클래스(자식클래스)에서 베이스클래스(부모클래스) 이름을 괄호로 삽입
- ◈ 복수의 부모 상속 가능 **(Multiple inheritance)**

```
class Animal:
    def _ _init_ _(self, name):
        self.name = name
    def move(self):
        print("move")
    def speak(self):
        pass

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

class Duck (Animal):
    def speak(self):
        print("quack")
```



```
dog = Dog("doggy") # 부모클래스의 생성자
n = dog.name # 부모클래스의 인스턴스변수
print "name= ", n
Print "name= ", dog.name
dog.move()   # 부모클래스의 메서드
dog.speak()  # 파생클래스의 멤버

결과)
name= doggy
Name=doggy
move
bark
```

3.7 입출력과 예외

- 파일로부터 데이터를 읽고 쓰기 위한 파일 객체
- 파일 읽기

```
f = open("test.txt")
data = f.readline()
while line:
    print line
    line = f.readline()
f.close()
```

- 파일 쓰기

```
f = open("out.txt", "w")
f.write("This file is %s" % ("out.txt"))
f.close()
```

◈ print 문으로 파일 지정 가능

```
f = open("out.txt", "w")
print >>f, "This file is %s" % ("out.txt"))
```

입출력과 예외

- 표준 입출력
  - sys 모듈 활용
- 표준입력 읽기
  - sys.stdin.readline() 또는 raw_input() 함수 사용

```
import sys
sys.stdout.write("Enter your name: ")
name = sys.stdin.readline()
```

```
name = raw_input()
```

- 표준 출력 쓰기
  - print 문 사용
  - format 함수 활용하면 효율적임

```
print format(year,"3d"),format(principal,"0.2f")
print(format(year,"3d"),format(principal,"0.2f")) # Python 3
```

```
print "{0:3d} {1:0.2f}".format(year,principal)
print("{0:3d} {1:0.2f}".format(year,principal)) # Python 3
```

입출력과 예외

pickle 모듈

- 객체의 내용을 바이트 스트림(byte stream)으로 직렬화하여 파일에 저장할 수 있도록 함
- dump(), load()함수를 사용하여 객체를 파일로 저장하거나 불러옴
- 예
  - 객체의 내용을 파일에 저장

```
import pickle
class TestObject(object):
    pass
o = TestObject()
f = open("test.pickle","wb")
pickle.dump(o,f)
f.close()
```

  - 객체의 내용을 파일로부터 불러옴

```
import pickle
f = open("test.pickle","rb")
o = pickle.load(f)
f.close()
```

입출력과 예외

▣ 예외 처리

- ◈ 프로그램 실행 도중 오류가 발생하면 관련된 예외 **(exception)** 가 발생하여 오류 메시지를 출력하고 프로그램이 종료
- ◈ 예외 처리 구문 사용 가능

```
try:
    f = open("test.txt", "r")
except IOError as e:
    print e
```

```
try:
   do something
except IOError as e:
   # Handle I/O error
   ...
except TypeError as e:
   # Handle Type error
   ...
except NameError as e:
   # Handle Name error
```

입출력과 예외

▣ 예외 처리

◈ 프로그램 실행 도중 오류가 발생하면

```
f = open('foo','r')
try:
    # Do some stuff
    ...
finally:
    f.close()
```

◈ 예외 발생

```
raise RuntimeError("Raise RuntimeError")
```

◈ 예외 처리 구문 사용 가능

```
import threading
message_lock = threading.Lock()
...
with message_lock:
    messages.add(newmessage)
```

3.8 모듈

- 파이썬 프로그램을 여러 조각으로 나누는 논리적인 방법
  - 관련된 변수, 함수, 클래스, 문장 등을 같은 이름을 가진 파일에 넣으면 됨
  - **import** 문을 써서 모듈을 포함하여 사용
  - 예

```
# mininum.py
def min(a,b):
    if a > b:
        return b
    else
        return a
```

```
import minimum
value = minimum.min(10, 20)
```

  - 모듈 이름 변경 가능

```
import minimum as Min
value = Min.min(10, 20)
```

모듈

◈ 모듈 내의 특정/전체 변수/함수를 가져올 때는 from 구문 사용사용

```
from minimum import min
from maximum import *
```

◈ 유용한 모듈들

이름	패키지명	설명
RPi.GPIO	python-rpi.gpio	라즈베리파이의 GPIO 핀 접근
webIOPi	webiopi	라즈베리파이를 위한 파이썬 웹서버 및 GPIO 제어
pygame	python-pygame	게임 프로그래밍 모듈
numpy	python-numpy	수학 및 과학계산용 기본 모듈
scipy	python-scipy	수학 및 과학계산용 모듈
PIL	python-imaging	이미지 처리 모듈
cv2	python-opencv	OpenCV 영상 처리 모듈
SimpleCV	없음	또다른 영상 처리 모듈
wxPython	python-wxgtk2.8	wxWidgets GUI 프로그래밍 모듈
Flask	python-flask	경량 웹 개발 프레임워크
Bottle	python-bottle	또다른 경량 웹 개발 프레임워크
PSerial	python-serial	직렬포트 접근