Hello, World

식별자 (Identifier)

- 변수, 함수, 클래스, 모듈 등을 구별하기 위해 사용하는 이름

- 명명 규칙
  - » 대소문자 구분
  - » 영문자, 숫자, _(underscores) 사용 가능
  - » 영문자 또는 _(underscores)로 시작
  - » 예약어를 사용할 수 없음

- 권장 명명 규칙
  - » 클래스 이름은 대문자로 시작하고 다른 식별자는 소문자로 시작
  - » 한 개의 언더스코어(_)로 시작하는 것은 private 변수
  - » 두 개의 언더스코어(__)로 시작하는 것은 강력한 private 변수 (이름 변형)
  - » 두 개의 언더스코어(__)로 시작하고, 끝나는 것은 언어에서 정의한 특별한 이름

예약어

- 특수한 목적으로 사용하는 이름 → 식별자로 사용할 수 없음

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

라인과 들여쓰기

- 파이썬은 클래스, 함수, 제어문 등의 코드 블록을 표시하기 위해 중괄호와 같은 특별한 표기를 사용하지 않음
  - » 코드 블록은 들여쓰기에 의해 구분되며 엄격하게 적용됨

- 사례

정상 사례

```
if True:
    print "True"
else:
  print "False"
```

비정상 사례

```
if True:
    print "Answer"
    print "True"
else:
    print "Answer"
  print "False"
```

다중행 실행문

- 파이썬은 한 줄에 하나의 명령문 작성이 원칙이나 예외적으로 여러 줄에 하나의 명령을 작성할 수 있는 방법 제공
  - » \를 이용한 연결

```
total = item_one + \
        item_two + \
        item_three
```

  - » [], {}, ()와 같은 목록 표시 내부에서는 여러 줄에 나누어서 작성 가능

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

- 한 줄에 2개 이상의 실행문 작성 → ; 사용

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

문자열과 주석

- 문자열을 표시하기 위해 ", ', """, 사용 가능
  - » 이 중 """은 여러 줄로 작성된 문자열 처리 지원

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

- 주석은 인터프리터가 해석하지 않는 영역 표시
  - » #으로 시작되며 그 줄 끝까지 주석

```
name = "Madisetti" # This is again comment
```

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

실행문 블록

- Suite → 여러 개의 개별 실행문이 하나의 실행문 블록을 만든 것

- if, while, def, class 등과 같은 복합 구문은 헤더 라인과 Suite 필요
  - » 헤더 라인은 : (콜론)으로 끝나고 Suite를 구성하는 한 개 이상의 라인이 이어짐

```
if expression :
   suite
elif expression :
   suite
else :
   suite
```

변수와 자료형

변수와 자료형

- 변수는 데이터를 저장하기 위해 예약된 메모리
- 자료형은 크기, 형식 등 데이터(변수)의 특성에 대한 설명
- 파이썬은 5개의 표준 자료형 제공
  - » Number, String, List, Tuple, Dictionary
- 변수의 자료형에 따라 인터프리터가 메모리를 할당하고 변수에 무엇이 저장될 수 있는지 결정
  - » 정수, 부동소수점, 문자열 등의 데이터를 구분해서 저장
  - » 자료형에 대한 명시적인 표기는 없으며 할당되는 데이터에 의해 판단

변수와 객체

- 객체는 프로그램으로 다루고자 하는 모든 것 또는 데이터이고 변수는 그 객체를 가리키는 것 (참조)

```
>>> a = 3
>>> b = 3
>>> a is b
True
```

a와 b는 같은 객체를 가르키는 서로 다른 참조

- 파이썬에서는 3, 1.1과 같은 단순 데이터도 객체로 취급

```
>>> a = 3
>>> type(a)
<class 'int'>
```

테스트 환경에 따라 다르게 표시될 수 있음

변수

- 변수 선언
  - » 변수에 값을 할당할 때 자동으로 변수가 만들어짐
  - » 할당할 때 = 연산자 사용

```
변수명 = 변수에 저장할 값
```

- 변수 선언 사례

```
counter = 100    # An integer assignment
miles = 1000.0   # A floating point
name = "John"    # A string

print(counter)
print(miles)
print(name)
```

```
100

1000.0

John
```

변수

- 하나의 값을 여러 변수에 할당

```
a = b = c = 1
```

- 여러 개의 값을 여러 변수에 할당

```
a, b, c = 1, 2, "John"
```

```
a, b, c = (1, 2, "John")
```

```
(a, b, c) = 1, 2, "John"
```

```
[a, b, c] = 1, 2, "John"
```

- 변수 제거 (자동으로 제거되기 때문에 명시적으로 삭제할 필요 없음)

```
a = 3
b = 3
del a
del b
```

숫자 자료형

- 수치 데이터를 저장하는 자료형

항목	사용 예
정수	123, -345, 0
실수	123.45, -1234.5, 3.4e10
8진수 (숫자0 + 영문자o로 시작)	0o34, 0o25
16진수 (숫자 0 + 영문자 x로 시작)	0x2A, 0xFF

- 수치 데이터 예제

정수형	실수형	8진수	16진수

`>>> a = 123`
`>>> a = -178`
`>>> a = 0`
	`>>> a = 1.2`
	`>>> a = -3.45`
	`>>> a = 4.24E10`
	`>>> a = 4.24e-10`
		`>>> a = 0o177`
		`>>> a = 0o234`
		`>>> a = 0o456`
			`>>> a = 0x8ff`
			`>>> a = 0xABC`
			`>>> a = 0x12D`

숫자 자료형

- 숫자 자료형 데이터를 처리하는 연산자

» 사칙연산 (+, -, *, /)

```
>>> a = 3
>>> b = 4
>>> a + b
7
>>> a * b
12
>>> a / b
0.75
```

» 나머지 연산 - 나눗셈의 나머지 반환 (%)

```
>>> 7 % 3
1
>>> 3 % 7
3
```

» 제곱연산 (**)

```
>>> a = 3
>>> b = 4
>>> a ** b
81
```

» 소수점 아래 버림 연산 (//)

```
>>> 7 // 3
2
>>> 3 // 7
0
```

숫자 자료형 형변환

- 숫자 자료형으로 형 변환을 처리하는 함수 제공

함수	설명
int(x)	x를 정수로 변환 (정수를 담은 문자열도 가능)
float(x)	x를 실수로 변환 (실수를 담은 문자열도 가능)

- 형변환 예제

```
>>> print( int(11.11) )
11

>>> print( float(11) )
11.0

>>> print( int("11") )
11

>>> print( float("11.11") )
11.11
```

문자 자료형

- 따옴표 내부에 표현된 연속된 문자 집합

- 작은 따옴표 또는 큰 따옴표 사용 가능
  - » 시작 따옴표와 끝 따옴표는 같은 형식의 따옴표 사용
  - » 3개의 따옴표를 연속으로 사용하면 여러 줄로 이루어진 문자열 생성 가능

```
"Hello World"
```

```
'Python is fun'
```

```
"""Life is too short, You need python"""
```

```
'''Life is too short, You need python'''
```

문자 자료형

- 문자열 연산
  - » 문자열 더하기 (연결)

```
>>> head = "Python"
>>> tail = " is fun!"
>>> head + tail
'Python is fun!'
```

  - » 문자열 곱하기 (반복)

```
>>> a = "python"
>>> a * 2
'pythonpython'
```

```
print("=" * 50)
print("My Program")
print("=" * 50)
```

```
==================================================
My Program
==================================================
```

문자 자료형

- 문자열 연산 계속
  - » 문자열 인덱싱과 슬라이싱
    - › [], [:] 연산자를 사용해서 문자열 내의 문자 또는 문자 집합을 추출
    - › 순서 번호는 0부터 시작
    - › 음수 인덱스는 뒤에서부터 시작

```
>>> a = "Life is too short, You need Python"
>>> a[3]
'e'
>>> a[0]
'L'
>>> a[12]
's'
>>> a[-1]
'n'
```

문자 자료형

- 문자열 연산 계속
  - » 문자열 인덱싱과 슬라이싱 계속

```
>>> a = "Life is too short, You need Python"
>>> a[0:4]
'Life'
>>> a[0:2]
'Li'
>>> a[5:7]
'is'
>>> a[12:17]
'short'
```

```
>>> a[19:]
'You need Python'
>>> a[:17]
'Life is too short'
>>> a[:]
'Life is too short, You need Python'
>>> a[19:-7]
'You need'
```

문자 자료형

- 문자열 포매팅
  - » 데이터와 문자열을 조합해서 새 문자열 생성할 때 사용
  - » 형식

```
"문자열과 서식 조합" % 단일 값 또는 튜플
```

```
"문자열과 { index } 조합".format(값 목록)
```

  - » 포맷 코드 : 문자열에 데이터를 채우기 위해 자료형에 따라 사용하는 표기법

코드	설명	코드	설명
%s	문자열 (String)	%o	8진수
%c	문자 1개(character)	%x	16진수
%d	정수 (Integer)	%%	Literal % (문자 % 자체)
%f	부동소수 (floating-point)		

    - › %s는 문자열 뿐만 아니라 다른 자료형에도 사용할 수 있음

문자 자료형

- 탈출 문자
  - » 문자열 내부에 enter, tab, backspace와 같은 특수한 문자를 표기하기 위해 미리 정의한 문자 집합

코드	설명	코드	설명
\n	개행 (줄바꿈)	₩r	캐리지 리턴
\t	수평 탭	\"	이중 인용부호(")
\\	문자 "₩"	\'	단일 인용부호(')
₩b	백 스페이스		

  - » 탈출 문자 예제

```
>>> multiline = "Life is too short!!!\n\tYou need \"python\""
>>> print(multiline)

Life is too short!!!
        You need "python"
```

문자 자료형

- 문자열 포매팅 예제

```
>>> "I eat %d apples." % 3
'I eat 3 apples.

>>> "I eat %s apples." % "five"
'I eat five apples.'

>>> number = 3
>>> "I eat %d apples." % number
'I eat 3 apples.'

>>> number = 10
>>> day = "three"
>>> "I ate %d apples. so I was sick for %s days." % (number, day)
'I ate 10 apples. so I was sick for three days.'
```

```
>>> "I have %s apples" % 3
'I have 3 apples'
>>> "rate is %s" % 3.234
'rate is 3.234'
```

%s는 문자열이 아닌 데이터에도 사용 가능

문자 자료형

- 문자열 포매팅 예제

```
>>> "I eat {0} apples".format(3)
'I eat 3 apples'

>>> "I eat {0} apples".format("five")
'I eat five apples’

>>> number = 3
>>> "I eat {0} apples".format(number)
'I eat 3 apples’

>>> number = 10
>>> day = "three"
>>> "I ate {0} apples. so I was sick for {1} days.".format(number, day)
'I ate 10 apples. so I was sick for three days.’

>>> "I ate {number} apples. so I was sick for {day}
days.".format(number=10, day=3)
'I ate 10 apples. so I was sick for 3 days.'
```

문자 자료형

- 문자열 포매팅 예제

```
>>> name = '홍길동'
>>> age = 30
>>> f'나의 이름은 {name}입니다. 나이는 {age}입니다.'
'나의 이름은 홍길동입니다. 나이는 30입니다.’

>>> age = 30
>>> f'나는 내년이면 {age+1}살이 됩니다.'
'나는 내년이면 31살이 됩니다.'

>>> d = {'name':'홍길동', 'age':30}
>>> f'나의 이름은 {d["name"]}입니다. 나이는 {d["age"]}입니다.'
'나의 이름은 홍길동입니다. 나이는 30입니다.’

>>> d = [ '홍길동', 30 ]
>>> f'나의 이름은 {d[0]}입니다. 나이는 {d[1]}입니다.'
'나의 이름은 홍길동입니다. 나이는 30입니다.'
```

리스트 자료형

- 다목적의 복합 데이터 타입
- [] 내부에 ,로 구분되는 항목들로 구성
- 서로 다른 자료형의 데이터를 목록에 포함할 수 있음
- 포함된 멤버들은 []와 [:] 연산자를 사용해서 접근
  - » 0부터 시작하는 인덱스 사용
  - » -(음수) 인덱스는 뒤에서부터 시작하는 위치 값
- + 연산자는 두 리스트를 결합
- * 연산자는 반복 연산 수행

리스트 자료형

- 리스트 만들기

```
>>> a = [ ] # a = list()
>>> b = [1, 2, 3]
>>> c = ['Life', 'is', 'too', 'short']
>>> d = [1, 2, 'Life', 'is']
>>> e = [1, 2, ['Life', 'is']]
```

- 리스트 인덱싱

```
>>> a = [1, 2, 3]
>>> a
[1, 2, 3]
>>> a[0]
1
>>> a[-1]
3
```

```
>>> a = [1, 2, 3, ['a', 'b', 'c']]
>>> a[0]
1
>>> a[-1]
['a', 'b', 'c']
>>> a[3]
['a', 'b', 'c']
>>> a[-1][1]
'b'
>>> a[-1][2]
'c'
```

리스트 자료형

- 리스트 슬라이싱

```
>>> a = [1, 2, 3, 4, 5]
>>> a[0:2]
[1, 2]
>>> a = "12345"
>>> a[0:2]
'12'
```

```
>>> a = [1, 2, 3, 4, 5]
>>> b = a[:2]
>>> c = a[2:]
>>> b
[1, 2]
>>> c
[3, 4, 5]
```

- 리스트 연산자

» + 연산자 (리스트 결합)

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a + b
[1, 2, 3, 4, 5, 6]
```

» * 연산자 (리스트 반복)

```
>>> a = [1, 2, 3]
>>> a * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

리스트 자료형

- 리스트 변경

» 단일 값 변경 (1)

```
>>> a = [1, 2, 3]
>>> a[2] = 4
>>> a
[1, 2, 4]
```

» 다중 값 변경 (2)

```
>>> a
[1, 2, 4]
>>> a[1:2]
[2]
>>> a[1:2] = ['a', 'b', 'c']
>>> a
[1, 'a', 'b', 'c', 4]
```

- 리스트 요소 삭제 (3)

```
>>> a
[1, 'a', 'b', 'c', 4]
>>> a[1:3] = [ ]
>>> a
[1, 'c', 4]
```

```
>>> a
[1, 'c', 4]
>>> del a[1]
>>> a
[1, 4]
```

리스트 자료형

- 리스트 관련 함수
  - » 리스트에 요소 추가

```
>>> a = [1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
```

```
>>> a.append([5,6])
>>> a
[1, 2, 3, 4, [5, 6]]
```

  - » 리스트 정렬

```
>>> a = [1, 4, 3, 2]
>>> a.sort()
>>> a
[1, 2, 3, 4]
```

```
>>> a = ['a', 'c', 'b']
>>> a.sort()
>>> a
['a', 'b', 'c']
```

  - » 리스트 뒤집기

```
>>> a = ['a', 'c', 'b']
>>> a.reverse()
>>> a
['b', 'c', 'a']
```

리스트 자료형

- 리스트 관련 함수
  - » 위치 반환

```
>>> a = [1,2,3]
>>> a.index(3)
2
>>> a.index(1)
0
```

  - » 리스트에 요소 삽입

```
>>> a = [1, 2, 3]
>>> a.insert(0, 4)
[4, 1, 2, 3]
```

```
>>> a.insert(3, 5)
[4, 1, 2, 5, 3]
```

  - » 리스트 요소 제거

```
>>> a = [1, 2, 3, 1, 2, 3]
>>> a.remove(3)
[1, 2, 1, 2, 3]
```

```
>>> a.remove(3)
[1, 2, 1, 2]
```

리스트 자료형

- 리스트 관련 함수
  - » 데이터 꺼내기 (읽기 + 삭제)

```
>>> a = [1,2,3]
>>> a.pop()
3
>>> a
[1, 2]
```

```
>>> a = [1,2,3]
>>> a.pop(1)
2
>>> a
[1, 3]
```

  - » 리스트에 포함된 요소 개수 세기

```
>>> a = [1,2,3,1]
>>> a.count(1)
2
```

  - » 리스트 확장

```
>>> a = [1,2,3]
>>> a.extend([4,5])
>>> a
[1, 2, 3, 4, 5]
```

```
>>> b = [6, 7]
>>> a.extend(b)
>>> a
[1, 2, 3, 4, 5, 6, 7]
```

튜플 자료형

- List와 비슷한 데이터 목록 자료형
- ()안에 ,로 구분되는 여러 개의 데이터로 구성됨
  - » 1개의 값을 갖는 경우 끝에 ,를 붙이지 않으면 단일 값 처리
- List는 요소와 크기를 변경할 수 있으나 Tuple은 변경 불가능 (읽기 전용 List)
- Tuple 예제

```
>>> t1 = ()
>>> t2 = (1,) # ,가 없으면 단일 값
>>> t3 = (1, 2, 3)
>>> t4 = 1, 2, 3
>>> t5 = ('a', 'b', ('ab', 'cd'))
```

튜플 자료형

- 튜플 값 삭제 → 오류

```
>>> t1 = (1, 2, 'a', 'b')
>>> del t1[0]
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

- 튜플 값 변경 → 오류

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0] = 'c'
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

튜플 자료형

- 튜플 인덱싱

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0]
1
>>> t1[3]
'b'
```

- 튜플 슬라이싱

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[1:]
(2, 'a', 'b')
```

- 튜플 더하기 (결합)과 튜플 곱하기 (반복)

```
>>> t2 = (3, 4)
>>> t1 + t2
(1, 2, 'a', 'b', 3, 4)
```

```
>>> t2 * 3
(3, 4, 3, 4, 3, 4)
```

딕셔너리 자료형

- key – value 세트로 구성되는 일종의 hash table 타입
  - » key로 대부분의 자료형을 사용할 수 있지만 일반적으로 숫자 또는 문자형 사용
  - » value로 모든 자료형의 데이터 사용

- {} 내부에 ,로 구분되는 key : value 세트의 목록으로 구성됨

- 항목에 접근할 때는 [] 사용

- 형식

```
{ Key1:Value1, Key2:Value2, Key3:Value3 ... }
```

딕셔너리 자료형

- 딕셔너리 만들기

```
>>> dic = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> a = {1: 'hi'}
>>> a = {'a': [1,2,3]}
```

- 딕셔너리 키:값 쌍 추가

```
>>> a = {1: 'a'}
>>> a[2] = 'b'
>>> a
{1: 'a', 2: 'b'}
>>> a['name'] = 'pey'
>>> a
{1: 'a', 2: 'b', 'name':'pey'}
>>> a[3] = [1,2,3]
>>> a
{1: 'a', 2: 'b', 'name':'pey', 3: [1, 2, 3]}
>>> a.update({1: 'modified', 4: 'appended'})
>>> a
{1: 'modified', 2: 'b', 'name':'pey', 3: [1, 2, 3], 4: 'appended'}
```

딕셔너리 자료형

- 딕셔너리 값 삭제

```
>>> del a[1]
>>> a
{'name': 'pey', 3: [1, 2, 3], 2: 'b'}
```

- 딕셔너리 사용 (키로 값 읽기)

  » 튜플과 리스트에 사용했던 인덱싱이나 슬라이싱은 사용할 수 없음

```
>>> grade = {'pey': 10, 'julliet': 99}
>>> grade['pey']
10
>>> grade['julliet']
99
>>> a = {1:'a', 2:'b'}
>>> a[1]
'a'
>>> a[2]
'b'
```

딕셔너리 자료형

- 딕셔너리 관련 함수
  - » Key 리스트 만들기
    - › keys() 함수로 dict_keys 형식의 값 반환 (2.7 버전에서는 리스트 반환)

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
>>> a.keys()
dict_keys(['name', 'phone', 'birth'])
```

  - » dict_keys 값 읽기

```
>>> for k in a.keys():
...     print(k)
...
phone
birth
name
```

  - » dict_keys를 리스트로 변환

```
>>> list(a.keys())
['phone', 'birth', 'name']
```

딕셔너리 자료형

- 딕셔너리 관련 함수
  - » Value 리스트 만들기
    - › values() 함수로 dict_valus 형식의 값 반환 (2.7 버전에서는 리스트 반환)

```
>>> a.values()
dict_values(['pey', '0119993323', '1118'])
```

  - » Key:Value 값 쌍 읽기
    - › items() 함수로 dict_items 형식의 값 반환

```
>>> a.items()
dict_items([('name', 'pey'), ('phone', '0119993323'), ('birth', '1118')])
```

  - » Key:Value 쌍 모두 지우기

```
>>> a.clear()
>>> a
{}
```

딕셔너리 자료형

- 딕셔너리 관련 함수
  - » Key로 Value 읽기 → get(key) 함수 사용

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> a.get('name')
'pey'
>>> a.get('phone')
'0119993323'
```

  - » Key로 Value 읽기 → get(key, value) : 키가 없을 경우 기본 값 사용

```
>>> a.get('foo', 'bar')
'bar'
```

  - » 특정 키가 딕셔너리에 있는지 조사 → in

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> 'name' in a
True
>>> 'email' in a
False
```

집합 자료형

- 집합에 관련된 것들을 쉽게 처리하기 위해 제공되는 형식
  - » 중복을 허용하지 않는 집합 데이터 형식
  - » 사용자가 데이터의 순서에 개입할 수 없는 데이터 형식
    - › 인덱싱을 통해 데이터 접근하려면 리스트나 튜플로 변경 후 사용

- set 키워드를 사용해서 생성

```
>>> s1 = set([1,2,3])
>>> s1
{1, 2, 3}
>>> s2 = set("Hello")
>>> s2
{'e', 'l', 'o', 'H'}
```

집합 자료형

- 집합 자료형 활용
  - » 교집합

```
>>> s1 = set([1, 2, 3, 4, 5, 6])
>>> s2 = set([4, 5, 6, 7, 8, 9])

>>> s1 & s2
{4, 5, 6}
>>> s1.intersection(s2)
{4, 5, 6}
```

  - » 합집합

```
>>> s1 | s2
{1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

집합 자료형

- 집합 자료형 활용
  - » 차집합

```
>>> s1 - s2
{1, 2, 3}
>>> s2 - s1
{8, 9, 7}
>>> s1.difference(s2)
{1, 2, 3}
>>> s2.difference(s1)
{8, 9, 7}
```

집합 자료형

- 집합 자료형 관련 함수
  - » 값 1개 추가

```
>>> s1 = set([1, 2, 3])
>>> s1.add(4)
>>> s1
{1, 2, 3, 4}
```

  - » 값 여러 개 추가

```
>>> s1 = set([1, 2, 3])
>>> s1.update([4, 5, 6])
>>> s1
{1, 2, 3, 4, 5, 6}
```

  - » 특정 값 제거

```
>>> s1 = set([1, 2, 3])
>>> s1.remove(2)
>>> s1
{1, 3}
```