

String and advanced function concept

TEAMLAB director

최성철

string

- 시퀀스 자료형으로 문자형 data를 메모리에 저장
- 영문자 한 글자는 1byte의 메모리공간을 사용

```
>>> import sys          # sys 모듈을 호출
>>> print (sys.getsizeof("a"), sys.getsizeof("ab"), sys.getsizeof("abc"))
50 51 52 # “ a ” , “ ab ” , “ abc ” 의 각 메모리 사이즈 출력
```

- string은 1byte 크기로 한 글자씩 메모리 공간이 할당됨

a = "abcde"

a	0100 1001
b	0100 1010
c	0100 1011
d	0100 1100
e	0100 1101

1Byte의 메모리 공간???

string

- 컴퓨터는 2진수로 데이터를 저장
- 이진수 한 자릿수는 1bit로 저장됨
- 즉 1bit 는 0 또는 1
- 1 byte = 8 bit = 2^8 = 256 까지 저장 가능

- 컴퓨터는 문자를 직접적으로 인식 X
→ 모든 데이터는 2진수로 인식
- 이를 위해 2진수를 문자로 변환하는 표준 규칙을 정함
- 이러한 규칙에 따라 문자를 2진수로 변환하여 저장하거나 저장된 2진수를 숫자로 변환하여 표시함
- 예) 대문자 U는 이진수로 "1000011" 변환됨 (UTF-8기준)

1Byte의 메모리 공간???

string

000	(nul)	016	► (dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	Ⓢ (soh)	017	◄ (dcl)	033	!	049	1	065	A	081	Q	097	a	113	q
002	Ⓢ (stx)	018	‡ (dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥ (etx)	019	!! (dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦ (eot)	020	℥ (dc4)	036	\$	052	4	068	D	084	T	100	d	116	t
005	♣ (enq)	021	§ (nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠ (ack)	022	— (syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	• (bel)	023	‡ (etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	▣ (bs)	024	↑ (can)	040	(056	8	072	H	088	X	104	h	120	x
009	(tab)	025	↓ (em)	041)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂ (vt)	027	← (esc)	043	+	059	;	075	K	091	[107	k	123	{
012	⌘ (np)	028	L (fs)	044	,	060	<	076	L	092	\	108	l	124	
013	(cr)	029	↔ (gs)	045	-	061	=	077	M	093]	109	m	125	}
014	♂ (so)	030	▲ (rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	⊗ (si)	031	▼ (us)	047	/	063	?	079	O	095	_	111	o	127	△

<http://goo.gl/K7Uz38>

- 각 타입 별로 메모리 공간을 할당 받은 크기가 다름

종류	타입	크기	표현 범위 (32bit)
정수형	int	4바이트	$-2^{31} \sim 2^{31}-1$
	long	무제한	무제한
실수형	float	8바이트	약 $10^{-308} \sim 10^{+308}$

<http://bit.ly/3pqkfa8>

- 메모리 공간에 따라 표현할 수 있는 숫자범위가 다름

예) 4byte = 32bit = $2^{32} = 4,294M = -2,147M \sim + 2.147M$ 까지 표시

- 데이터 타입은 메모리의 효율적 활용을 위해 매우 중요

- 문자열의 각 문자는 개별 주소(offset)를 가짐
- 이 주소를 사용해 할당된 값을 가져오는 것이 인덱싱
- List와 같은 형태로 데이터를 처리함

```
>>> a = "abcde"
>>> print (a[0], a[4])      # a 변수의 0번째, 4번째 주소에 있는 값
a e
>>> print (a[-1], a[-5])   # a 변수의 오른쪽에서 0번째, 4번째 주소에 있는 값
e a
```

Hello

0 1 2 3 4
-5 -4 -3 -2 -1

각 문자의 오프셋은

왼쪽에선 0부터 오른쪽에선 -1부터 시작함

```
>>> a = "abcde"
>>> print (a[0], a[4])      # a 변수의 0번째, 4번째 주소에 있는 값
a e
>>> print (a[-1], a[-5])   # a 변수의 오른쪽에서 0번째, 4번째 주소에 있는 값
e a
```

- 문자열의 주소값을 기반으로 문자열의 부분값을 반환

```
>>> a = "Artificial Intelligence and Machine Learning"
>>> print (a[0:6], " AND ", a[-9:]) # a 변수의 0부터 5까지, -9부터 끝까지
Artifi AND Learning
>>> print (a[:]) # a변수의 처음부터 끝까지
Artificial Intelligence and Machine Learning
>>> print (a[-50:50]) # 범위를 넘어갈 경우 자동으로 최대 범위를 지정
Artificial Intelligence and Machine Learning
>>> print (a[::2], " AND ", a[::-1]) # 2칸 단위로, 역으로 슬라이싱
Atfca nelgneadMcieLann AND gninrael enihcaM dna ecnegilletnI laicifitrA
```

- 덧셈과 뺄셈 연산 가능, in 명령으로 포함여부 체크

```
>>> a = "TEAM"
>>> b = "LAB"
>>> print (a + " " + b)           # 덧셈으로 a와 b 변수 연결하기
TEAM LAB
>>> print (a * 2 + " " + b * 2)
TEAMTEAM LABLAB # 곱하기로 반복 연산 가능
>>> if 'A' in a: # 'U' 가 a에 포함되었는지 확인
...     print (a)
... else:
...     print (b)
...
TEAM
```

문자열 함수 (1/2)

string

함수명	기능
<code>len(a)</code>	문자열의 문자 개수를 반환
<code>a.upper()</code>	대문자로 변환
<code>a.lower()</code>	소문자로 변환
<code>a.capitalize()</code>	첫 문자를 대문자로 변환
<code>a.title()</code>	제목형태로 변환 띄워쓰기 후 첫 글자만 대문자
<code>a.count('abc')</code>	문자열 a에 'abc'가 들어간 횟수 반환
<code>a.find('abc')</code> <code>a.rfind('abc')</code>	문자열 a에 'abc'가 들어간 위치(오프셋) 반환
<code>a.startswith('abc')</code>	문자열 a는 'abc'로 시작하는 문자열여부 반환
<code>a.endswith('abc')</code>	문자열 a는 'abc'로 끝나는 문자열여부 반환

문자열 함수 (2/2)

string

함수명	기능
<code>a.strip()</code>	좌우 공백을 없앴
<code>a.rstrip()</code>	오른쪽 공백을 없앴
<code>a.lstrip()</code>	왼쪽 공백을 없앴
<code>a.split()</code>	공백을 기준으로 나눠 리스트로 반환
<code>a.split('abc')</code>	abc를 기준으로 나눠 리스트로 반환
<code>a.isdigit()</code>	문자열이 숫자인지 여부 반환
<code>a.islower()</code>	문자열이 소문자인지 여부 반환
<code>a.isupper()</code>	문자열이 대문자인지 여부 반환

문자열 함수 예시

string

```
>>> title = "TEAMLAB X Upstage"
>>> title.upper()
'TEAMLAB X UPSTAGE'
>>> title.lower()
'teamlab x upstage'
>>> title.split(" ")
['TEAMLAB', 'X', 'Upstage']
>>> title.isdigit()
False
>>> title.title()
'Teamlab X Upstage'
>>> title.startswith("a")
False
>>> title.count("a")
1
>>> title.upper().count("a")
0
```

```
>>> "12345".isdigit()
True
>>> title.find(" Naver")
0
>>> title.upper().find(" Naver")
-1
>>> "    Hello    ".strip()
'Hello'
>>> "A-B-C-D-E-F".split("-")
['A', 'B', 'C', 'D', 'E', 'F']
```

문자열 선언은 큰따옴표(“)나 작은 따옴표 (')를 활용

It's OK 이라는 문자열은 어떻게 표현할까?

① a = 'It\W' ok.'

\W'는 문자열 구분자가 아닌 출력 문자로 처리

② a = "It's ok." #

큰따옴표로 문자열 선언 후 작은 따옴표는 출력 문자로 사용

두줄 이상은 어떻게 저장할까?

① \n # \n은 줄바꿈을 의미하는 특수 문자

② 큰따옴표 또는 작은 따옴표 세 번 연속 사용

예) a= """ It'Ok

I'm Happy.

See you. """

문자열을 표시할 때 백슬래시 “\” 를 사용하여 키보드로 표시하기 어려운 문자들을 표현함

문자	설명	문자	설명
\ [Enter]	다음 줄과 연속임을 표현	\n	줄 바꾸기
<u>\\</u>	\\ 문자 자체	\t	TAB 키
\`	` 문자	\e	ESC 키
\"	" 문자		
\b	백 스페이스		

특수문자 특수 기호인 \ escape 글자를 무시하고 그대로 출력함

```
>>> raw_string = "이제 파이썬 강의 그만 만들고 싶다. \n 레알"
```

```
>>> print(raw_string)
```

이제 파이썬 강의 그만 만들고 싶다.

레알

```
>>> raw_string = r"이제 파이썬 강의 그만 만들고 싶다. \n 레알"
```

```
>>> print(raw_string)
```

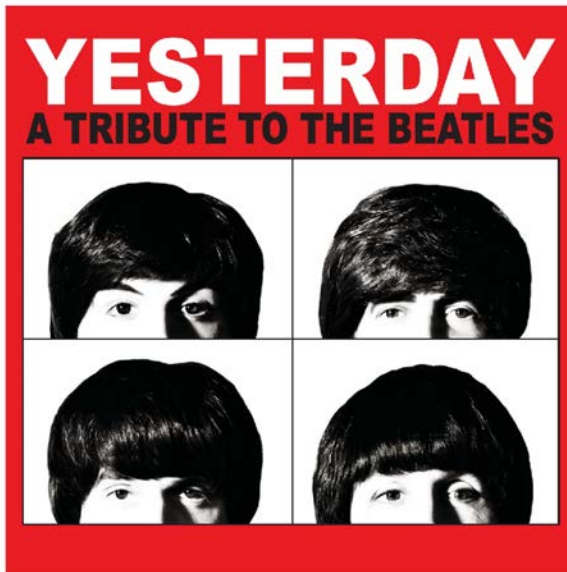
이제 파이썬 강의 그만 만들고 싶다. \n 레알

str lab - yesterday

Yesterday 노래엔 Yesterday 라는 말이 몇 번 나올까?

wget

https://raw.githubusercontent.com/TeamLab/introduction_to_python_TEAMLAB_MOOC/master/code/6/yesterday.txt



Yesterday 노래엔 Yesterday 라는 말이 몇 번 나올까?

```
f = open("yesterday.txt", 'r')
yesterday_lyric = ""
while True:
    line = f.readline()
    if not line:
        break
    yesterday_lyric = yesterday_lyric + line.strip() + "\n"
f.close()
n_of_yesterday = yesterday_lyric.upper().count("YESTERDAY") # 대소문자 구분 제거
print ("Number of a Word 'Yesterday'" , n_of_yesterday)
```

Yesterday 노래엔 Yesterday 라는 말이 몇 번 나올까?
대소문자를 구분하여 "Yesterday"와 "yesterday"의
개수를 나눠서 세는 프로그램을 작성하세요.

function II

call by object reference

함수에서 parameter를 전달하는 방식

- (1) 값에 의한 호출(Call by Value)
- (2) 참조의 의한 호출(Call by Reference)
- (3) 객체 참조에 의한 호출(Call by Object Reference)

Call by Value

함수에 인자를 넘길 때 값만 넘김.

함수 내에 인자 값 변경 시, 호출자에게 영향을 주지 않음

Call by Reference

함수에 인자를 넘길 때 메모리 주소를 넘김.

함수 내에 인자 값 변경 시, 호출자의 값도 변경됨

파이썬은 객체의 주소가 함수로 전달되는 방식

전달된 객체를 참조하여 변경 시 호출자에게 영향을 주나,
새로운 객체를 만들 경우 호출자에게 영향을 주지 않음

```
def spam(eggs):  
    eggs.append(1) # 기존 객체의 주소값에 [1] 추가  
    eggs = [2, 3] # 새로운 객체 생성  
  
ham = [0]  
spam(ham)  
print(ham) # [0, 1]
```

- 함수를 통해 변수 간의 값을 교환(Swap)하는 함수
- Call By XXXX를 설명하기 위한 전통적인 함수 예시

```
Enter the value of x and y
4
5
Before Swapping
x = 4
y = 5
After Swapping
x = 5
y = 4
```

a = [1,2,3,4,5] 일 때 아래 함수 중 실제 swap이 일어나는 함수는?

```
def swap_value (x, y):  
    temp = x  
    x = y  
    y = temp
```

```
def swap_offset (offset_x, offset_y):  
    temp = a[offset_x]  
    a[offset_x] = a[offset_y]  
    a[offset_y] = temp
```

```
def swap_reference (list, offset_x, offset_y):  
    temp = list[offset_x]  
    list[offset_x] = list[offset_y]  
    list[offset_y] = temp
```

swap_offset: a 리스트의 전역 변수 값을 직접 변경

swap_reference: a 리스트 객체의 주소 값을 받아 값을 변경

```
a = [1,2,3,4,5]
swap_value(a[1], a[2])
print (a)                # [1,2,3,4,5]
swap_offset(1,2)
print (a)                # [1,3,2,4,5]
swap_reference(a, 1, 2)
print (a)                # [1,3,2,4,5]
```

function – scoping rule

- 변수가 사용되는 범위 (함수 또는 메인 프로그램)
- 지역변수(local variable) : 함수내에서만 사용
- 전역변수(Global variable) : 프로그램전체에서 사용

```
def test(t):  
    print(x)  
    t = 20  
    print ("In Function :", t)
```

```
x = 10  
test(x)  
print(t)
```

Function Call Test

function

```
def test(t):  
    t = 20  
    print ("In Function :", t)  
  
x = 10  
print ("Before :", x)    # 10  
test(x)                  # 함수 호출  
print ("After :", x)     # 10 - 함수 내부의 t는 새로운 주소값을 가짐
```

- 전역변수는 함수에서 사용가능
- But, 함수 내에 전역 변수와 같은 이름의 변수를 선언하면 새로운 지역 변수가 생김

```
def f():  
    s = "I love London!"  
    print(s)  
  
s = "I love Paris!"  
f()  
print(s)
```

- 함수 내에서 전역변수 사용 시 global 키워드 사용

```
def f():  
    global s  
    s = "I love London!"  
    print(s)  
  
s = "I love Paris!"  
f()  
print(s)
```

변수의 범위 (Scoping Rule) - test

function - scoping rule

```
def calculate(x, y):  
    total = x + y          # 새로운 값이 할당되어 함수 내 total은 지역변수가 됨  
    print ("In Function")  
    print ("a:", str(a), "b:", str(b), "a+b:", str(a+b), "total :", str(total))  
    return total  
  
a = 5                      # a와 b는 전역변수  
b = 7  
total = 0                  # 전역변수 total  
print ("In Program - 1")  
print ("a:", str(a), "b:", str(b), "a+b:", str(a+b))  
  
sum = calculate (a,b)  
print ("After Calculation")  
print ("Total :", str(total), " Sum:", str(sum)) # 지역변수는 전역변수에 영향 X
```

recursive function

- 자기자신을 호출하는 함수
- 점화식과 같은 재귀적 수학 모형을 표현할 때 사용
- 재귀 종료 조건 존재, 종료 조건까지 함수호출 반복

$$n! = n \cdot (n - 1) \cdots 2 \cdot 1 = \prod_{i=1}^n i$$

$$1! = 1$$

$$2! = 2(1) = 2$$

$$3! = 3(2)(1) = 6$$

$$4! = 4(3)(2)(1) = 24$$

$$5! = 5(4)(3)(2)(1) = 120$$

- 자기자신을 호출하는 함수
- 점화식과 같은 재귀적 수학 모형을 표현할 때 사용
- 재귀 종료 조건 존재, 종료 조건까지 함수호출 반복

```
def factorial(n):  
    if n == 1:  
        return 1  
    else:  
        return n + factorial(n-1)  
  
print (factorial(int(input("Input Number for Factorial Calculation: "))))
```


연습문제 - 아래 재귀함수 코드를 일반 loop 코드로 변경

```
def factorial(n):  
    if n == 1:  
        return 1  
    else:  
        return n + factorial(n-1)  
print (factorial(int(input("Input Number for Factorial Calculation: "))))
```

function type hints

- 파이썬의 가장 큰 특징 - dynamic typing
 - 처음 함수를 사용하는 사용자가 interface를 알기 어렵다는 단점이 있음
- python 3.5 버전 이후로는 PEP 484에 기반하여 type hints 기능 제공

```
def do_function(var_name: var_type) -> return_type:  
    pass
```

```
def type_hint_example(name: str) -> str:  
    return f"Hello, {name}"
```

- Type hints의 장점

- (1) 사용자에게 인터페이스를 명확히 알려줄 수 있다.
- (2) 함수의 문서화시 parameter에 대한 정보를 명확히 알 수 있다.
- (3) mypy 또는 IDE, linter 등을 통해 코드의 발생 가능한 오류를 사전에 확인
- (4) 시스템 전체적인 안정성을 확보할 수 있다.

```
def insert(self, index: int, module: Module) -> None:
    r"""Insert a given module before a given index in the list.
    Arguments:
        index (int): index to insert.
        module (nn.Module): module to insert
    """
    for i in range(len(self._modules), index, -1):
        self._modules[str(i)] = self._modules[str(i - 1)]
    self._modules[str(index)] = module
```

<https://github.com/pytorch/pytorch/blob/master/torch/nn/modules/container.py>

docstring

- 파이썬 함수에 대한 상세스펙을 사전에 작성 → 함수 사용자의 이행도 UP
- 세개의 따옴표로 docstring 영역 표시(함수명 아래)

```
def kos_root():  
    """Return the pathname of the KOS root directory."""  
    global _kos_root  
    if _kos_root: return _kos_root  
    ...
```

<https://www.datacamp.com/community/tutorials/docstrings-python>

```
def add_binary(a, b):                                     https://www.programiz.com/python-programming/docstrings
    """
    Returns the sum of two decimal numbers in binary digits.

    Parameters:
        a (int): A decimal integer
        b (int): Another decimal integer

    Returns:
        binary_sum (str): Binary string of the sum of a and b
    """
    binary_sum = bin(a+b)[2:]
    return binary_sum

print(add_binary.__doc__)
```


최근에는 black 모듈을 활용하여 pep8 like 수준을 준수
black codename.py 명령을 사용

```
D:\workspace\ai-pnpp\codes\python (main -> origin)
(upstage) λ black function_hints.py
reformatted function_hints.py
All done! ✨ 💡 ✨
1 file reformatted.
```

함수 개발 가이드라인

- 함수는 가능하면 짧게 작성할 것 (줄 수를 줄일 것)
- 함수 이름에 함수의 역할, 의도가 명확히 들어날 것

```
def print_hello_world():  
    print("Hello, World")  
  
def get_hello_world():  
    return "Hello, World")
```

- 하나의 함수에는 유사한 역할을 하는 코드만 포함

```
def add_variables(x,y):  
    return x + y
```

```
def add_variables(x,y):  
    print (x, y)  
    return x + y
```

- 인자로 받은 값 자체를 바꾸진 말 것 (임시변수 선언)

```
def count_word(string_variable):  
    string_variable = list(string_variable)  
    return len(string_variable)
```



```
def count_word(string_variable):  
    return len(string_variable)
```

- 공통적으로 사용되는 코드는 함수로 변환
- 복잡한 수식 → 식별 가능한 이름의 함수로 변환
- 복잡한 조건 → 식별 가능한 이름의 함수로 변환

```
a = 5
if (a > 3):
    print "Hello World"
    print "Hello Gachon"
if (a > 4):
    print "Hello World"
    print "Hello Gachon"
if (a > 5):
    print "Hello World"
    print "Hello Gachon"
```



```
def print_hello():
    print "Hello World"
    print "Hello Gachon"

a = 5
if (a > 3):
    helloMethod()

if (a > 4):
    helloMethod()

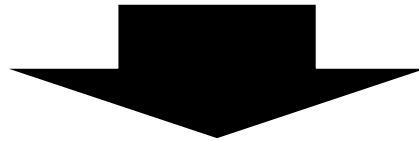
if (a > 5):
    helloMethod()
```

복잡한 수식은 함수로

How to write good code

```
import math
a = 1; b = -2; c = 1

print ((-b + math.sqrt(b ** 2 - (4 * a * c))) / (2 * a))
print ((-b - math.sqrt(b ** 2 - (4 * a * c))) / (2 * a))
```



```
import math

def get_result_quadratic_equation(a, b, c):
    values = []
    values.append((-b + math.sqrt(b ** 2 - (4 * a * c))) / (2 * a))
    values.append((-b - math.sqrt(b ** 2 - (4 * a * c))) / (2 * a))
    return values

print (get_result_quadratic_equation(1,-2,1))
```


How to write Good Code

코딩은 팀플레이





컴퓨터가 이해할 수 있는
코드는 어느 바보나 다 짤 수 있다.

좋은 프로그래머는 사람이
이해할 수 있는 코드를 짤다.

- 마틴 파울러 -

사람의 이해를 돕기 위해 규칙이 필요함

우리는 그 규칙을
코딩 컨벤션
이라고 함

- 명확한 규칙은 없음
- 때로는 팀마다, 프로젝트마다 따로
- 중요한 건 **일관성!!!**
- 읽기 좋은 코드가 좋은 코드

- 들여쓰기는 **Tab or 4 Space** 논쟁!
- 일반적으로 4 Space를 권장함
- 중요한 건 **혼합하지 않으면 됨**

- 들여쓰기 공백 4칸을 권장
- 한 줄은 최대 79자까지
- 불필요한 공백은 피함

```
def factorial( n ):
    if n == 1:
        return 1
```


- = 연산자는 1칸 이상 안 띄움

```
variable_example = 12  
variable_example = 12
```

- 주석은 항상 갱신, 불필요한 주석은 삭제
- 코드의 마지막에는 항상 한 줄 추가

- 소문자 l, 대문자 O, 대문자 I 금지

```
lIOO = "Hard to Understand"
```

- 함수명은 소문자로 구성, 필요하면 밑줄로 나눔

"flake8" 모듈로 체크 - flake8 <파일명>

conda install -c anaconda flake8

```
lL00 = "123"  
for i in 10 :  
    print ("Hello")
```

```
flake8 flake8_test.py  
flake8_test.py:2:12: E203 whitespace before ':'  
flake8_test.py:3:10: E211 whitespace before '('
```

End of Document
Thank You.