Lecture #04 | 자료구조와 반복문 1: `list`

SE213 프로그래밍 (2019)

Written by Mingyu Cho

Edited by Donghoon Shin

지난 시간에 다룬 내용

- 함수
  - 함수 소개
  - 함수의 정의와 인자 전달 방법
  - 네임스페이스와 지역 변수
  - `return`문과 반환값
  - lambda 함수

오늘 다룰 내용

- 전역, 지역 변수
- 자료구조
  - 리스트(list)
- 반복문
  - while
  - for
- 함수의 인자와 반환값으로 리스트 사용

함수, 변수 이름

- 함수명, 변수명 규칙(python 2.x)
  - 첫 글자: 알파벳 문자 혹은 밑줄(_)
  - 나머지 글자: 문자, 밑줄, 숫자
  - 대/소문자를 구별
  - python keyword는 제외
- (변경) python 3.x
  - A 에서 Z 범위의 대문자와 소문자, 밑줄 _, 첫 문자를 제외하고, 숫자 0 에서 9
  - (추가) ASCII 범위 밖의 문자(PEP 3131 참조)
    - unicodedata 모듈에 포함된 버전의 유니코드 문자
    - https://www.dcl.hpi.uni-potsdam.de/home/loewis/table-3131.html

[Recap] 함수의 정의와 사용

- 함수 정의

```
def function_name(param1, param2, param3=default_value):
    ''' this is an example function '''
    statement1
    ...
    return return_value
```

- 함수 사용

```
var = function_name(arg1, arg2, param3=keyword_arg)
```

[Recap] 지역 변수과 변수의 범위(scope)

```
def foo(x):
    x = 28

def bar(y):
    y = 496

x = 6
y = 42
foo(x)
bar(x)
bar(y)
print(x, y)
```

Local Scope

Local Scope

Global Scope

```
6 42
```

전역 변수, 지역 변수

- 함수에서 전역 변수 설정

```
var = 40
def fn ():
    global var
```

- 중첩 함수에서 지역 변수 설정

```
def outer_fn ():
    var = 40
    def inner_fn ():
        nonlocal var
```

지역 변수과 변수의 범위(scope)

```
def scope_test():
    def do_local():
        var = "local var"

    def do_nonlocal():
        nonlocal var
        var = "nonlocal var"

    def do_global():
        global var
        var = "global var"

    var = "test var"
    do_local()
    print("After local assignment:", var)
    do_nonlocal()
    print("After nonlocal assignment:", var)
    do_global()
    print ("After global assignment:", var)

scope_test()
print("In global scope:", var)
```

```
After local assignment: test var
After nonlocal assignment: nonlocal var
After global assignment: nonlocal var
In global scope: global var
```

‘프로그래밍’ 교과에서 지금까지 다룬 내용...?

- 데이터
  - 자료형: int, float, str, …
  - 변수
  - ***자료구조***
- 연산
  - 연산자: +, −, *, /, ...
  - 조건문
  - 함수
  - ***반복문***

자료구조 (Data structure)

- 자료 구조: 자료를 구조화하고 저장하는 방식
  - 하나의 변수에 하나 이상의 값 혹은 객체를 저장하기 위하여 주로 사용됨
- python은 특성이 다른 여러 형태의 자료 구조를 제공함
  - 리스트(`list`)
  - 튜플(`tuple`)
  - 사전(`dict`)
  - 집합(`set`)

시퀀스(sequence)

- 시퀀스는 python의 자료 구조의 종류의 하나임, 예
  - 변경 가능한 시퀀스(mutable): list
  - 변경 불가능한 시퀀스(immutable): tuple, str
- 공통점
  - 유한한 순서가 있는 집합으로, 음이 아닌 정수로 그 순서(index)를 나타냄
  - 슬라이싱(slicing, 시퀀스의 부분집합)을 제공
  - 유사한 형태의 여러 연산자 혹은 함수를 제공함

리스트(list)

- 정의: *mutable* sequences, typically used to store collections of homogeneous items
  - mutable: 원소의 추가/삭제 혹은 값이 변경될 수 있음
  - 원소의 자료형: 임의의 자료형의 원소를 저장할 수 있음
- 인덱스(index): 음이 아닌 정수로 원소의 순서를 나타냄
- 표기법: [] 를 사용하고, 원소는 , 로 구분
- 수학에서 행렬/벡터 혹은 C/C++/Java의 array와 유사함

list의 정의

- 리스트를 정의할 때, [ ]를 사용하고, 각 원소는 ,로 구분한다

```
# defining lists
t1 = [42, 1024, 23] # a list with 3 elements
print(t1)

t2 = [] # empty list
print(t2)
```

```
[42, 1024, 23]
[]
```

list의 원소 접근

- 리스트의 원소는 0부터 시작하는 인덱스로 접근할 수 있다
- 음수가 사용되면, 가장 마지막 원소부터 역순으로 접근한다

```
t1 = [42, 1024, 23]
print(t1[0])
print(t1[1])
t1[2] = 7
print(t1[2])
print(t1)
print(t1[-1])
print(t1[-2])
print(t1[-3])
```

```
42
1024
7
[42, 1024, 7]
7
1024
42
```

list의 여러 원소들 접근: 슬라이싱(slicing)

- *name_of_list*[*start*:*end*]
  - *name_of_list*의 [*start*, *end*)에 위치한 원소들을 가지는 리스트를 생성
  - *start*: 생략되면 첫 원소부터
  - *end*: 생략되면 마지막 원소까지

```
t = [42, 1024, 23, 6, 28, 496]
print(t)
print(t[1:4])
print(t[3:])
print(t[:2])
print(t[:])  # copy of a list*
```

```
[42, 1024, 23, 6, 28, 496]
[1024, 23, 6]
[6, 28, 496]
[42, 1024]
[42, 1024, 23, 6, 28, 496]
```

* To be discussed later with more details

list에 대한 연산*

```
t1 = [42, 1024, 23]
print(1024 in t1)
print(7 in t1)
print(1024 not in t1)
print(len(t1)) # the number of elements
print(min(t1)) # the minimum value
print(max(t1)) # the maximum value
t2 = [6, 28, 496]
t3 = t1 + t2 # concatenation
print(t3)
t4 = t2 * 2 # repetition
print(t4)
```

```
True
False
False
3
23
1024
[42, 1024, 23, 6, 28, 496]
[6, 28, 496, 6, 28, 496]
```

* 여기 소개된 대부분의 연산이 다른 시퀀스(str, tuple)에도 적용됨

list에 원소 추가하기

- `append()`: list에 원소를 하나씩 추가
- `extend()`: 한 list에 다른 list의 모든 원소를 추가

```
t1 = []
t1.append(42)
t1.append(1024)
t1.append(23)
print(t1)
t2 = [6, 28, 496]
# equivalent to t1 += t2
t1.extend(t2)
print(t1)
```

```
[42, 1024, 23]
[42, 1024, 23, 6, 28, 496]
```

반복문

- while
  - 어떤 조건이 만족되는 동안 반복
  - 주로 반복회수를 모를 때 사용
    - 예: 조건에 맞거나 맞지 않을 때까지 사용자/파일/네트웍 입력 등에 사용
- for
  - 시퀀스의 모든 원소에 대해서 반복을 수행
  - 주로 얼마나 반복을 해야되는지 알 경우에 사용
    - 예: 시퀀스의 있는 모든 원소들에 대한 작업 수행, n번 반복
- 참고: while과 for를 모두 쓸 수 있는 경우, for가 선호되는 경우가 많음

제어 흐름: while

```
while condition:
    statement1
    statement2
else:
    statement3
    statement4
```

A: statement1, statement2

B: statement3, statement4

- *condition*이 True일 때
  - A 부분 명령어를 수행
  - 다시 *condition* 계산
  - 참고: *condition*이 True인 경우 A 부분을 계속 수행
- else절 (else-clause)
  - *condition*이 False일 때, B부분의 코드블럭 수행
  - 생략 가능

예제: while

```
counter = 0
while counter < 3:
    print(counter)
    counter += 1

# similar to the following
counter = 0
print(counter)
counter += 1 # counter = 1
print(counter)
counter += 1 # counter = 2
print(counter)
counter += 1 # counter = 3
```

```
0
1
2
```

제어 흐름: for

```
for variable in sequence:
    statement1
    statement2
else:
    statement3
    statement4
```

A: statement1, statement2

B: statement3, statement4

- *sequence*에 있는 원소들을 순서대로 *variable*에 대입한 후, A 부분 명령어를 수행
  → *sequence*에 있는 원소 각각에 대해 수행할 연산을 정의함
- *sequence*에 저장된 모든 원소들에 대해 연산을 수행한 후, else 이후 B 부분 명령어를 수행 (else절은 생략 가능)

예제: for

```
# a simple for loop
for value in [42, 1024, 23]:
    print(value)

# similar to the following
value = 42
print(value)
value = 1024
print(value)
value = 23
print(value)
```

```
42
1024
23
```

range()

- Range()
  - range(*stop*) : immutable sequence i (0 <= i < stop)
  - range(*start*, *stop*[, *step*]) : immutable sequence i (start <= i < stop, i += step)

```
# range() example
print(range(5))
print(list(range(5)))
print(list(range(1, 4)))
print(list(range(1, 4, 2)))
print(list(range(1, 5, 2)))

print(list(range(1, -5, -2)))
print(list(range(1, 5, -2)))
```

```
range(0, 5)
[0, 1, 2, 3, 4]
[1, 2, 3]
[1, 3]
[1, 3]
[1, -1, -3]
[]
```

예제: for loop with range() as a sequence

- 특정 회수를 반복하거나 특정 정수의 범위에 대한 반복을 하기 위해 사용
- range() 함수는 리스트를 반환한다고 생각해도 프로그램 실행을 이해하는 것에는 무방함

```
# using range() with for loop
for index in range(5):
    print(index)



for index in range(3, 10, 3):
    print(index, end='  ')
```

```
0
1
2
3
4
3 6 9
```

예제: for loop with range() and len()

- 리스트의 인덱스를 이용한 반복을 하기 위해서 자주 사용하는 구문
- 리스트의 각 원소의 값을 변경하는 경우 주로 사용됨

```
t = [42, 1024, 23]
for i in range(len(t)):
    print('t[', i, '] = ', t[i], sep='')
    t[i] = t[i] * 2

print(t)
```

```
t[0] = 42
t[1] = 1024
t[2] = 23
[84, 2048, 46]
```

함수를 호출 할 때, 인자로 리스트를 전달

```
def display(sequence):
    for v in sequence:
        print(v, end=' ')
    print()


t1 = [42, 1024, 23]
t2 = [6, 28, 496]
display(t1)
display(t2)
```

```
42 1024 23
6 28 496
```

함수의 반환값으로 리스트를 사용

```
import random

def random_list(n, k):
    new_list = []
    for i in range(k):
        new_list.append(random.randrange(n))
    return new_list


t = random_list(5, 2)
print(t)
print(random_list(10, 3))
```

```
[3, 4]
[5, 2, 3]
```

읽을 거리

- Python tutor: http://pythontutor.com
- Why numbering should start at zero by Edsger Dijkstra: http://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD831.html

ANY QUESTIONS?