Lecture #05 | 자료구조와 반복문 2: list, tuple

SE213 프로그래밍 (2018)

지난 시간에 다룬 내용

- 자료구조
  - 리스트(list)
  - 문자열(str)
- 반복문
  - while
  - for
- 함수의 인자와 반환값으로 리스트 사용

오늘 다룰 내용

- 자료구조
  - 리스트(list): 추가 내용
  - 튜플(tuple)
  - 튜플 언팩킹
- 반복문 추가 설명
  - for의 추가적인 용법
  - break, continue
- pass
- __name__

리스트가 지원하는 함수 (일부)

- `list.insert(i, x)`: Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.
- `list.remove(x)`: Remove the first item from the list whose value is *x*. It is an error if there is no such item.
- `list.clear()`: Remove all items from the list. Equivalent to `del a[:]`.
- `list.sort(key=None, reverse=False)`: Sort the items of the list in place (the arguments can be used for sort customization, see `sorted()` for their explanation).
- `list.reverse()`: Reverse the elements of the list in place.
- `list.copy()`: Return a shallow copy of the list. Equivalent to `a[:]`.

* https://docs.python.org/3/tutorial/datastructures.html

예시: 리스트의 함수들 (추가)

```
t1 = [42, 1024, 23]
print(t1)
t1.reverse()
print(t1)
t1.sort()
print(t1)
t1.insert(1, 6)
print(t1)
t1.remove(42)
print(t1)
t1.sort(reverse=True)
print(t1)
t1.clear()
print(t1)
```

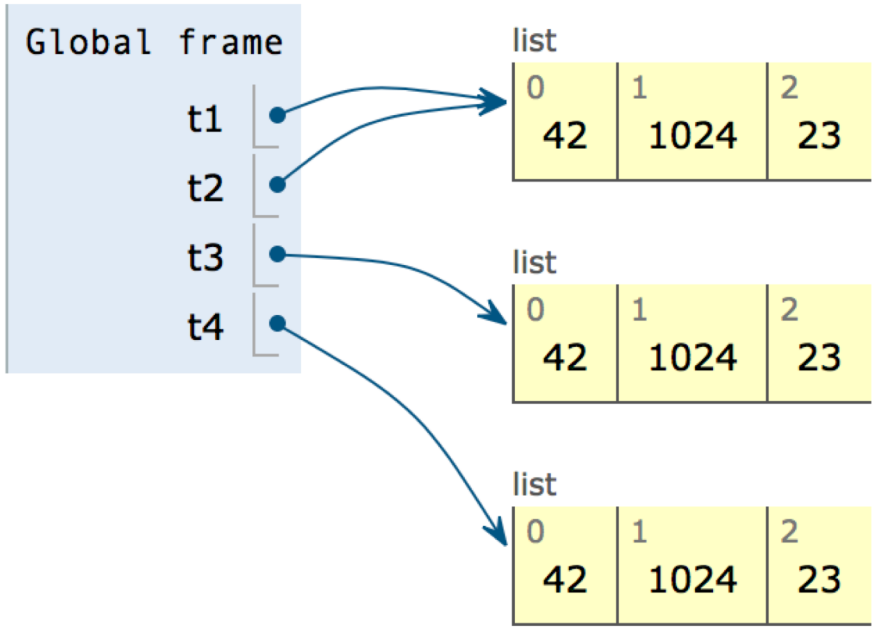
```
[42, 1024, 23]
[23, 1024, 42]
[23, 42, 1024]
[23, 6, 42, 1024]
[23, 6, 1024]
[1024, 23, 6]
[]
```

예시: 리스트의 복사, copy() 함수

```
t1 = [42, 1024, 23]
t2 = t1
t3 = t1.copy() # copy t1 (make a new list)
t4 = t1[:] # copy t1 (make a new list)

t2[2] = 6
t3[2] = 28
t4[2] = 496
print(t1, t2, t3, t4, sep='\n')
```

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



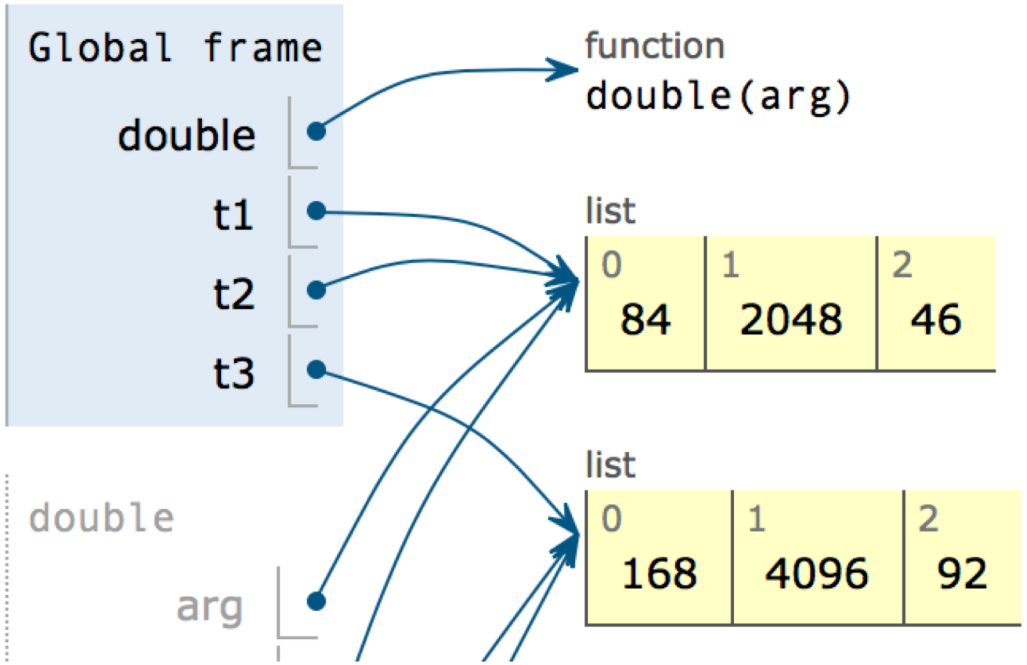
* 오른쪽은 4행까지 실행한 후의 상태

예시: 함수 인자로 리스트 사용

```
def double(arg):
    for i in range(len(arg)):
        arg[i] = arg[i] * 2
    return arg


t1 = [42, 1024, 23]
t2 = double(t1)
t3 = double(t1.copy())
t2[2] = 6
t3[2] = 28
print(t1, t2, t3, sep='\n')
```

```
[84, 2048, 6]
[84, 2048, 6]
[168, 4096, 28]
```



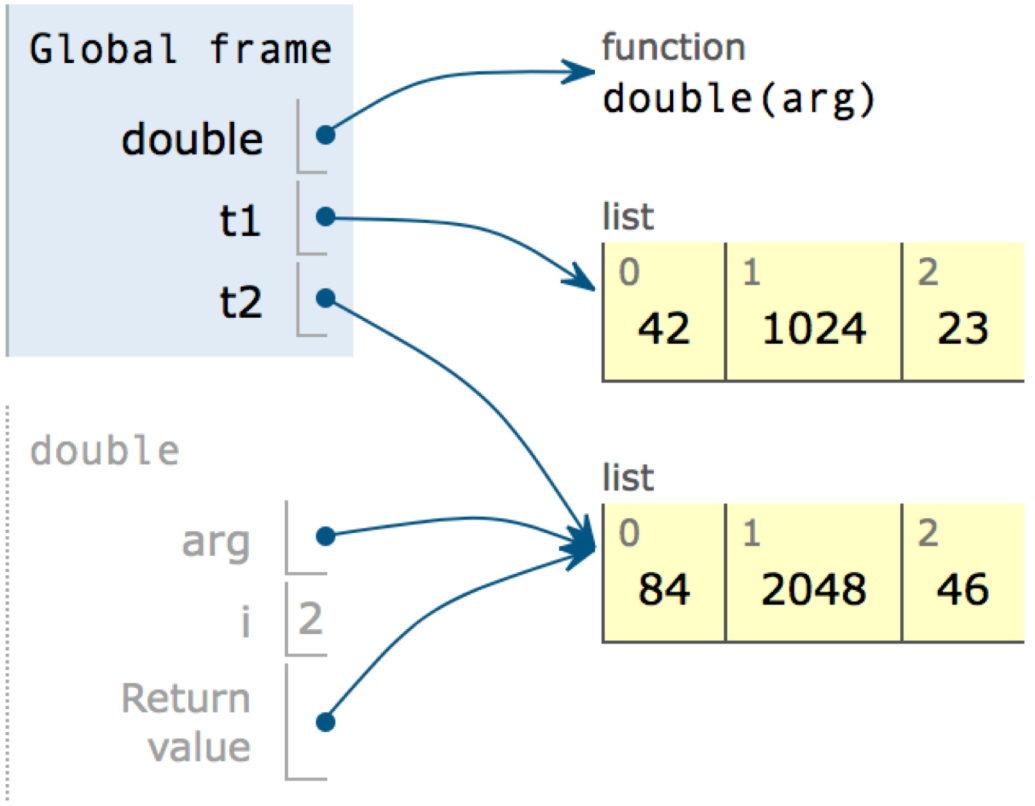
* 오른쪽 상태는 마지막 3줄을 실행하기 전의 상황

예시: 함수 인자로 리스트 사용 (cont.)

```
def double(arg):
    arg = arg.copy()
    for i in range(len(arg)):
        arg[i] = arg[i] * 2
    return arg


t1 = [42, 1024, 23]
t2 = double(t1)
print(t1)
print(t2)
```

```
[42, 1024, 23]
[84, 2048, 46]
```



* 오른쪽 상태는 마지막 2줄을 실행하기 전의 상황

튜플(tuple)

- 정의: *immutable* sequences, typically used to store collections of homogeneous items
  - immutable: 원소의 추가/삭제, 값의 변경이 불가능
  - 원소의 자료형: 임의의 자료형의 원소를 저장할 수 있음
- 인덱스(index): 음이 아닌 정수로 원소의 순서를 나타냄
- 표기법: () 를 사용하고, 원소는 ,로 구분
- 리스트와 유사한 점
  - 인덱싱
  - (원소의 변경이 없는) 연산 혹은 함수

tuple에 대한 연산

```
t1 = (42, 1024, 6)
t2 = (42, ) # tuple with 1 element
print(t1[1])
print(t1[1:3])
print(len(t1))
print(min(t1))
print(max(t1))
t2 = t1 + t1
t3 = t1 * 3
print(t2)
print(t3)
t1[2] = 3 # error
t2.append(0) # error
```

```
1024
(1024, 6)
3
6
1024
(42, 1024, 6, 42, 1024, 6)
(42, 1024, 6, 42, 1024, 6, 42, 1024, 6)
```

튜플 언패킹 (tuple unpacking)

- 튜플에 있는 값들을 변수에 대입하는 것을 튜플 언패킹이라 함
  - 참고: 튜플을 정의하는 것을 튜플 패킹(tuple packing)이라 함
  - 튜플 패킹/언패킹 때 ()는 생략 가능함
- 튜플 언패킹 때, 원소의 개수와 변수의 개수가 일치해야 함*
- 참고: 다른 시퀀스(`list`, `str` 등)에서도 언패킹이 지원됨

```
my_tuple = (42, 1024, 23)  # tuple packing
(first, second, third) = my_tuple  # tuple unpacking
print('first = ', first, ', second = ', second, ', third = ', third, sep='')
```

```
first = 42, second = 1024, third = 23
```

* 가변 길이의 원소를 언패킹하는 방법이 python3부터 지원됨

예시: 튜플/리스트/스트링 언패킹

```
my_tuple = 42, 1024, 23
first, second, third = my_tuple
print(my_tuple)
print('first = ', first, ', second = ', second, ',
third = ', third, sep='')
print(type(my_tuple))
my_list = [42, 1024, 23]
(a, b, c) = my_list
print('a = ', a, ', b = ', b, ', c = ', c, sep='')
string = 'Wow'
(a, b, c) = string
print('a = ', a, ', b = ', b, ', c = ', c, sep='')
```

```
(42, 1024, 23)
first = 42, second = 1024,
third = 23
<class 'tuple'>
a = 42, b = 1024, c = 23
a = W, b = o, c = w
```

예시: 변수의 값 교환 (swap)

```
"""Swap two value
1. make a tuple (var2, var1) with values of var2 and var1
2. unpack the tuple to var1 and var2
"""
var1 = 42
var2 = 1024
var1, var2 = var2, var1
print(var1, var2)
```

```
1024 42
```

예시: 함수의 반환값으로 튜플 사용

```
def divide(dividend, divisor):
    """Return (quotient, remainder)"""
    quotient = dividend // divisor
    remainder = dividend % divisor
    return quotient, remainder # return a tuple



q, r = divide(13, 3) # tuple unpacking
print('quotient = ', q, ', remainder = ', r, sep='')
```

```
quotient = 4, remainder = 1
```

list, tuple, str의 비교

- list: mutable sequences
- tuple: immutable sequences
- str: immutable sequences of Unicode code points
  - immutable: 변경이 불가능
  - Unicode: 한중일 등 세계 각국 문자를 포함하는 문자표현 방식 (python 3부터 적용)
- list와 str은 sequence로 유사한 연산들이 정의되어 있음

Recap: 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]
```

예제: for loop with enumerate()

- for문 안에 인덱스와 값을 동시에 필요한 경우가 있음 → enumerate() 사용
- enumerate(thing)은 인덱스와 값의 튜플들을 반환함, 예를 들어,
  - (0, thing[0]), (1, thing[1]), (2, thing[2]), ...

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

print(t)
```

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

예제: 중첩된 반복문 – nested for loops

```
for v1 in [42, 1024]:
    print(v1)
    for v2 in [6, 23]:
        print(v1 + v2)
```

```
v1 = 42
print(v1)
v2 = 6
print(v1 + v2)
v2 = 23
print(v1 + v2)
v1 = 1024
print(v1)
v2 = 6
print(v1 + v2)
v2 = 23
print(v1 + v2)
```

```
42
48
65
1024
1030
1047
```

제어 흐름: break, continue, pass

- break: 가장 안쪽의 for 혹은 while문의 실행을 끝냄
  - break문을 이용해 반복문의 실행을 끝내면 else절이 실행되지 않음
- continue: loop의 다음 반복을 수행
  - while: 조건이 True이면 while문 내부의 코드블럭 실행
  - for: sequence의 다음 값을 loop variable에 대입하고 for문 내부 코드블럭 실행
- pass: 어떤 연산도 수행하지 않고, 주로 문법적으로 필요한 경우 사용됨 (placeholder)

예제: else절이 있는 for문

- for/while문의 else절은 for/while문의 반복이 끝난 뒤 실행됨
  - for: sequence의 모든 값들을 loop variable에 대입한 후
  - while: 조건이 False가 된 이후

```
t = [42, 1024, 23]
for v in t:
    print(v)
else:
    print('end of for loop')
```

```
42
1024
23
end of for loop
```

예제: break

```
number = 7
for i in range(2, number):
    if number % i == 0:
        break
else:
    print(number, 'is a prime number.')
```

```
7 is a prime number.
```

예제: continue

```
for num in range(2, 10):
    if num % 2 == 0:
        print('Found an even number', num)
        continue
    print('Found a number', num)
```

```
Found an even number 2
Found a number 3
Found an even number 4
Found a number 5
Found an even number 6
Found a number 7
Found an even number 8
Found a number 9
```

예제: pass

```
# pass is typically used as a placeholder
def foo():
    pass

for i in range(10):
    pass

while False:
    pass
```

예제: __name__

- __name__: Python script (주로 확장자가 .py인 파일)이 실행될 때, 어떤 이름으로 실행되는지를 나타내는 변수
  - 독립된 변수로 실행될 때: '__main__'
  - import 됐을 때: 모듈 이름 (즉 파일 이름)*
- 이 코드가 필요한 이유
  - import 되지 않았을 때만 실행하는 코드를 위해서 사용함
  - 주로 코드에 포함된 클래스/함수의 테스트 혹은 예제 코드를 포함하게 됨

```
if __name__ == '__main__':
    some_statements
```

* 나중에 다시 다룰 내용

ANY QUESTIONS?