Lecture #06 | 자료구조와 반복문: 추가 설명 문자열 처리

SE213 프로그래밍 (2019)

Written by Mingyu Cho
Edited by Donghoon Shin

지난 시간에 다룬 내용

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

오늘 다룰 내용

- 복합자료구조
- 반복문
  - while v.s. for
  - while문 추가 설명
- 문자열 처리
  - 문자열 서식화

1st assignment

- 과제 1-1: 윤년 확인
  - 윤년을 확인하는 함수 leap_year()를 작성하시오.
    - 서력 기원 연수가 4로 나누어떨어지는 해는 윤년으로 한다. (1988년, 1992년, 1996년, 2004년, 2008년, 2012년 …)
    - 이 중에서 100으로 나누어떨어지는 해는 평년으로 한다. (1900년, 2100년, 2200년, 2300년, 2500년 …)
    - 그중에 400으로 나누어떨어지는 해는 윤년으로 둔다. (1600년, 2000년, 2400년 …)
- 과제 1-2: 세금과 팁의 계산
  - 함수 vat(), tip(), total()를 작성하시오.
    - vat : 음식가격과, vat 세율을 인자로 받아서 vat 금액 리턴 (소수점 이하 2자리)
    - tip : 음식가격, vat 세율, 팁 %, 세후/세전 값을 인자로 받아서 tip 금액 리턴 (소수점 이하 2자리)
    - total : 음식가격, vat 세율, 팁 %, 세후/세전 값을 인자로 받아서 총 지불해야 하는 금액 리턴 (소수점 이하 2자리)
- 과제 1-3: 올림픽 메달리스트
  - 반복문을 이용하여 8명 선수의 점수를 입력 받고 scores(리스트 타입)에 각각 저장하시오.
  - 함수 print_medalists()를 작성하시오.
    - 선수이름(players)과 점수(scores)를 인자로 받아서 화면에 높은 점수 순으로 금메달, 은메달, 동메달 선수를 출력하는 코드를 작성하고 반환값은 없음

round 함수

- round 함수: 인자로 설정한 정확도 만큼 반올림 결과를 리턴

round(*number*[, *ndigits*])

Return *number* rounded to *ndigits* precision after the decimal point. If *ndigits* is omitted or is `None`, it returns the nearest integer to its input.

For the built-in types supporting `round()`, values are rounded to the closest multiple of 10 to the power minus *ndigits*; if two multiples are equally close, rounding is done toward the even choice (so, for example, both `round(0.5)` and `round(-0.5)` are `0`, and `round(1.5)` is `2`). Any integer value is valid for *ndigits* (positive, zero, or negative). The return value is an integer if *ndigits* is omitted or `None`. Otherwise the return value has the same type as *number*.

For a general Python object `number`, `round` delegates to `number.__round__`.

https://docs.python.org/3/library/functions.html#round

리스트의 원소로 리스트/튜플 사용

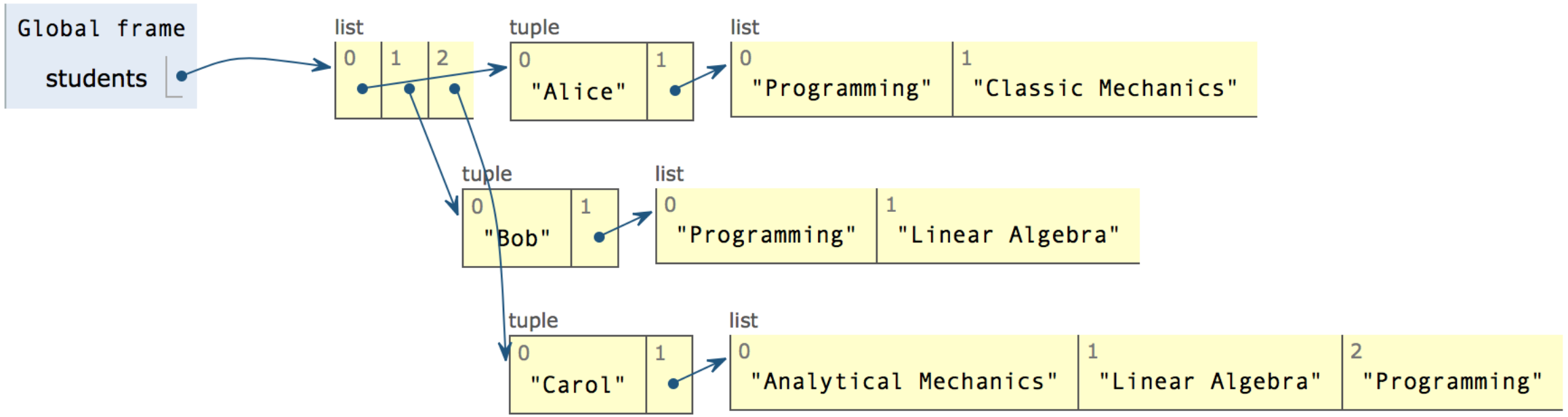
- 리스트는 원소로 python의 어떠한 데이터형도 가질 수 있음
  → 기본자료형 뿐만 아니라 리스트, 튜플 등을 원소로 가질 수 있음

```
students = [
    ('Alice', ['Programming', 'Classic Mechanics']),
    ('Bob', ['Programming', 'Linear Algebra']),
    ('Carol', ['Analytical Mechanics', 'Linear Algebra', 'Programming'])]
print(students[2])
print(students[2][1])
print(students[2][1][2])
```

```
('Carol', ['Analytical Mechanics', 'Linear Algebra', 'Programming'])
['Analytical Mechanics', 'Linear Algebra', 'Programming']
Programming
```

참고: 리스트의 원소로 리스트/튜플 사용한 경우 메모리 상태

```
students = [
    ('Alice', ['Programming', 'Classic Mechanics']),
    ('Bob', ['Programming', 'Linear Algebra']),
    ('Carol', ['Analytical Mechanics', 'Linear Algebra', 'Programming'])
```



예시: 리스트의 원소가 리스트/튜플일 때 반복문

```
students = [
    ('Alice', ['Programming', 'Classic Mechanics']),
    ('Bob', ['Programming', 'Linear Algebra']),
    ('Carol', ['Analytical Mechanics', 'Linear Algebra', 'Programming'])]

for student in students:
    print('Name :', student[0])
    print('Courses :', student[1])
```

```
Name : Alice
Courses : ['Programming', 'Classic Mechanics']
Name : Bob
Courses : ['Programming', 'Linear Algebra']
Name : Carol
Courses : ['Analytical Mechanics', 'Linear Algebra', 'Programming']
```

예시: 리스트의 원소가 리스트/튜플일 때 반복문 (cont.)

```
students = [
    ('Alice', ['Programming', 'Classic Mechanics']),
    ('Bob', ['Programming', 'Linear Algebra']),
    ('Carol', ['Analytical Mechanics', 'Linear Algebra', 'Programming'])]

for student_name, courses in students:
    print('Name :', student_name)
    print('Courses :', courses)
```

```
Name : Alice
Courses : ['Programming', 'Classic Mechanics']
Name : Bob
Courses : ['Programming', 'Linear Algebra']
Name : Carol
Courses : ['Analytical Mechanics', 'Linear Algebra', 'Programming']
```

가변 언패킹(unpacking)

- 튜플/리스트에 있는 값들을 각각 변수에 대입하는 것을 언패킹이라 함
- 튜플 언패킹 때, 원소의 개수와 변수의 개수가 일치해야 함*
  - 단, python3 부터는 가변 길이 원소 언패킹 가능

```
my_list = [1, 2, 3, 4]
first, second, third, forth = my_list  # unpacking


first, *rest = my_list
first, second, *rest = my_list
*rest, last = my_list
first, *rest, last = my_list
```

예시: 리스트의 원소가 리스트/튜플일 때 반복문 (cont.)

```
students = [
    ('Alice', ['Programming', 'Classic Mechanics']),
    ('Bob', ['Programming', 'Linear Algebra']),
    ('Carol', ['Analytical Mechanics', 'Linear Algebra', 'Programming'])]

for student_name, (course1, course2, *rest) in students:
    print('Name :', student_name)
    print('Course :', course1)
```

```
Name : Alice
Course : Programming
Name : Bob
Course : Programming
Name : Carol
Course : Analytical Mechanics
```

Recap: 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

Recap : 중첩된 반복문 - nested for loops

```
for outer in range(2):
    for inner in range(3):
        print('start')
        print('end')
```

```
outer = 0
for inner in range(3):
    print('start')
    print('end')

outer = 1
for inner in range(3):
    print('start')
    print('end')
```

```
outer = 0
inner = 0
print('start')
print('end')
inner = 1
print('start')
print('end')
inner = 2
print('start')
print('end')
outer = 1
for inner in
range(3):
    print('start')
    print('end')
```

Recap : break, continue

```
for i in range(3):
    print('start')
    if i == 1 :
        break
    print('end')
```

```
i = 0
print('start')
if i == 1 :
    break
print('end')
```

```
i = 1
print('start')
if i == 1 :
    break
print('end')
```

```
i = 2
print('start')
if i == 1 :
    break
print('end')
```

```
i = 0
print('start')
if i == 1 :
    continue
print('end')
```

```
i = 1
print('start')
if i == 1 :
    continue
print('end')
```

```
i = 2
print('start')
if i == 1 :
    continue
print('end')
```

예제: while v.s. for – 0에서 2까지 출력하기

- 반복되는 회수/범위가 명확할 때는 for문을 쓰는 것이 바람직한 경우가 많음
  - 가독성이 좋을 뿐더라, 실수를 하는 것을 방지해줌
  - while문은 같은 기능을 하는 코드를 여러 가지 방법으로 구현하는 것이 가능함

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

```
counter = 0
while counter <= 2:
    print(counter)
    counter += 1
```

```
counter = -1
while counter <= 1:
    counter += 1
    print(counter)
```

```
counter = -1
while counter < 2:
    counter += 1
    print(counter)
```

```
for counter in range(3):
    print(counter)
```

예제: Lab #04의 연습 3

- input_numbers()
  - 사용자로부터 입력받은 정수들의 값들을 리스트에 넣어 반환하는 함수
  - 이 때, 입력받은 값이 0이면 더 이상 입력을 받지 않음

```
def input_numbers():
  numbers = []
  number = int(input())
  while number != 0:
   if number > 0:
     numbers.append(number)
   number = int(input())

  return numbers
```

```
def input_numbers():
  numbers = []
  number = 1
  while number != 0:
   number = int(input())
   if number > 0:
     numbers.append(number)

  return numbers
```

```
def input_numbers():
  numbers = []
  while True:
   number = int(input())
   if number == 0:
     break
   elif number > 0:
     numbers.append(number)

  return numbers
```

for 반복문 - 무한 반복

```
numbers = [1, 2, 3]
for number in numbers:
    print(number)
    if number % 2 == 0:
        numbers.append(number)

print(numbers)
```

```
1
2
3
2
2
...
...
...
```

```
numbers = [1, 2, 3]
for number in numbers[:]:
    print(number)
    if number % 2 == 0:
        numbers.append(number)

print(numbers)
```

```
1
2
3
[1, 2, 3, 2]
```

문자열 서식화 (string formatting)

- python에서 문자열 서식화
  - 서식문자열에 따라, (변수에 저장된) 값의 자리수, 표현방식을 서식화함
  - 서식화된 '**문자열**'을 반환 → 변수에 대입하거나, `print()` 함수를 이용하여 출력할 수 있음
    - 참고: C언어의 `printf()` 함수가 아닌, `sprintf()` 함수에 해당
- python에서 지원하는 문자열 서식화 방식
  - **Printf-style string formatting** (all python versions)
    - C언어의 `printf()` 함수와 유사학 서식문자열을 지원
  - `format()` **function with format string syntax** (python 3, python 2.6+)
    - 서식문자열의 표현이 더 풍부해짐
  - **Literal string interpolation** (python 3.6+)
    - 서식문자열에 수식 혹은 변수의 이름을 적을 수 있는 기능을 추가
  - **Template string** (python 2+): printf-style과 유사

printf-style string formatting (% operator*)

- 사용형태: `format % values`
  - `format`: 서식 문자열로, 다음 형태의 변환명시자(conversion specifier)를 포함
    - %로 시작하고, 변환형을 나타내는 하나의 문자로 끝남
    - mapping key, conversion flag, 최소 문자 폭, 정확도를 명시할 수 있음
  - `values`: 다음 중 하나의 형식
    - 한 개의 값 (서식 문자열이 하나의 값을 요구할 때)
    - 서식문자열이 요구하는 아이템 수와 값은 수의 값이 있는 튜플
    - 한 개의 사전(dict)과 같은 매핑 객체 (mapping object)
- 기능: 서식 문자열에 따라 `values`에 주어진 값들을 원하는 형식의 **'문자열'**로 변환함
- 단점: 튜플, 사전, 객체 등 복잡한 자료형의 값을 출력할 때 제한점이 많음
- 참고: C언어의 `printf()` 혹은 `sprintf()` 함수에서 서식문자열과 유사함

* string formatting or interpolation operator라고도 불림

변환 명시자 (conversion specifier) 형식

1. %로 시작
2. (Optional) mapping key
   - ()안에 문자들로 주어짐
   - 맵핑 객체 형태의 values에서 키가 ()안의 문자들인 값으로 대체됨
3. (Optional) conversion flag (일부)
   - '0': 수의 경우, 남는 공간이 0으로 채워짐
   - '-': 변환된 문자열이 왼쪽에 정렬됨
4. (Optional) 최소 문자 폭: 출력되는 값의 최소 문자의 수를 지정하는 정수
5. (Optional) 정확도: '.'와 숫자로, 소수 부분의 자리수을 명시
6. (Optional) length modifier: python에서는 사용되지 않음
7. 변환형을 나타내는 하나의 문자로 끝남 (다음 페이지 참고)

Reference: 변환형

Conversion	Meaning
'd' or 'i'	Signed integer decimal.
'o'	Signed octal value.
'u'	Obsolete type – it is identical to 'd'.
'x' or 'X'	Signed hexadecimal (lowercase ('x') or uppercase('X').
'e' or 'E'	Floating point exponential format (lowercase ('e') or uppercase('E').
'f' or 'F'	Floating point decimal format.
'g' or 'G'	Floating point format. Uses lowercase ('g') or uppercase ('G') exponential format if exponent is less than -4 or not less than precision, decimal format otherwise.
'c'	Single character (accepts integer or single character string).
'r', 's', 'a'	String (converts any Python object using repr(), str(), and ascii(), respectively)
'%'	No argument is converted, results in a '%' character in the result.

예시: printf-style string formatting

```
var = 42
string = 'Answer'
pi = 3.141592653589793
s1 = 'The %s is %d.' % (string, var)
print(s1)
print('0123456789' * 2)
print('%10d' % var)
print('%010d' % var)
print('%-10d' % var)
print('%f' % pi)
print('%.2f' % pi)
print('%.10f' % pi)
print('%10.2f' % pi)
print('%g, %g, %g' % (var, pi, 10000000000))
print('%(name)s: %(average)g' %
        {'name': 'Alice', 'average': 99.9})
```

```
The Answer is 42.
01234567890123456789
        42
0000000042
42        
3.141593
3.14
3.1415926536
      3.14
42, 3.14159, 1e+10
Alice: 99.9
```

* 빈칸은 ␣로 표시함

format() 함수를 이용한 문자열 서식화

- 사용형태: string.format(arguments)
  - string: 서식 문자열
  - arguments: 서식 문자열에 의해 변환된 인자들
- 서식 문자열에 '{fieldname:format_spec}'의 형태가 주어지면 인자의 값의 서식을 변경함
  - fieldname: 어떤 인자의 값을 서식화할 지를 지정
  - format_spec: 어떠한 형태로 서식화할 지를 지정
    - [[fill]align] [sign] [0] [width] [.precision] [type]
- 참고: 서식화를 지정하는 서식 명시자(formatting specifier)는 printf 형태의 변환 명시자와 유사하나, 그 기능이 확장되었고 사용이 조금 더 편리하도록 변경함

예시: format() 함수

```
d = 42
f = 3.14
s = 'apple'
print('{} {} {}'.format(d, f, s))
print('{2} {0} {1}'.format(d, f, s))
print('{0} is {0}'.format(s))
print('{d} {f} {s}'.format(d=6,
f=1.618, s='pineapple'))
print('{0:d} {0:f}
{0:g}'.format(42))
```

```
42 3.14 apple
apple 42 3.14
apple is apple
6 1.618 pineapple
42 42.000000 42
```

예시: format() 함수 (cont.)

```
print('0123456789' * 2)
print('{:10d}'.format(42))
print('{:>10d}'.format(42))
print('{:<10d}'.format(42))
print('{:^10d}'.format(42))
print('{:10.2f}'.format(3.1415926535))
print('{:1d}'.format(42))
print('{:5.5f}'.format(3.1415926535))
```

```
01234567890123456789
        42
        42
42
    42
      3.14
42
3.14159
```

* 빈칸은 ␣로 표시함

Formatted string literal을 이용한 문자열 서식화

- f-string
  - 문자열 앞에 f를 이용하여 표시*
  - 서식 표현은 `format()` 함수와 유사
  - `'{}'` 안에 계산이 가능한 표현(expression)이 들어감
  - f-string이 실행될 때, `'{}'` 안의 표현이 연산(evaluate)됨

* python에서 문자열 앞에 한 글자를 추가하여, 특수한 문자열을 나타냄

예시: Literal string interpolation

```
pi = 3.14159265
width = 10
precision = 2
print('0123456789' * 3)
print(f'{pi}')
print(f'{pi:10.2f}')
print(f'{pi:{width}.{precision}f}')
print(f'{max(width, precision)}')
t = [42, 1024, 23]
for i in range(len(t)):
    print(f'{i}: {t[i]}')
```

```
012345678901234567890123456789
3.14159265
␣␣␣␣␣␣3.14
␣␣␣␣␣␣3.14
10
0:␣42
1:␣1024
2:␣23
```

* 빈칸은 ␣로 표시함

읽을 거리

- python 문자열 자료형과 함수: https://wikidocs.net/13
- python 문자열 서식화
  - PyFormat Using % and .format() for great good!: https://pyformat.info/
  - The 4 Major Ways to Do String Formatting in Python: https://dbader.org/blog/python-string-formatting
- References
  - printf-style string formatting: https://docs.python.org/3/library/stdtypes.html?highlight=printf#old-string-formatting
  - Format string syntax: https://docs.python.org/3/library/string.html#formatstrings
  - Literal string interpolation: https://www.python.org/dev/peps/pep-0498/

ANY QUESTIONS?