Lecture #02 | 변수, 입력, 조건문, 수학 함수 사용

SE213 프로그래밍 (2018)

지난 시간에 다룬 내용

- 강좌 소개 및 운영에 관한 사항
- 프로그래밍 과목을 배우는 이유 혹은 기대할 것
- 컴퓨터 혹은 프로그래밍은 무엇인가?
- python소개와 기초 문법
  - 기본 자료형과 연산
  - 변수
  - 표준 출력: `print()`

오늘 다룰 내용

- 변수
  - 변수 대입문과 축약형
  - 변수의 저장공간
- 표준 입력(standard input): `input()`
- 형변환 (type conversion)
- 조건문: `if`, `else`, `elif`
- 주석
- 모듈 사용: `math`, `random`

[Recap] 변수

- 변수가 필요한 이유
  - 동일한 값을 여러 번 사용할 필요가 있음
  - 중간 결과를 나타내기 위해서 사용함
  - 수학에서 사용하는 변수와 유사함
- 정의
  - 어떠한 값(value) 혹은 객체(object)에 대한 이름(alias)
  - 값 혹은 객체를 저장할 수 있는 이름이 붙어있는 메모리 상의 공간
- 변수에 값 혹은 객체를 대입(assignment)한 이후, 대입된 값 혹은 객체 대신에 변수의 이름을 사용할 수 있음

1 https://docs.python.org/3/reference/lexical_analysis.html?#keywords

[Recap] 대입(assignment)

- *variable_name = expression*
  - 의미: 오른쪽 expression의 결과값을 왼쪽 변수에 대입하는 명령
    - 주의: 변수 이름이 항상 왼쪽에 있어야 함
    - 참고: 등식은 ==로 나타냄
  - 참고: 일부 언어에서 대입문으로 사용하는 기호: :=, <-
- 예제

```
greeting = 'Hello, world!'
n = 42
pi = 3.14159265
r = 3.0
```

대입에서 축약형

- 축약형: 변수의 값을 이용해 연산을 하고, 다시 그 값을 변수에 대입하는 대입연산자
  - 형태: *variable #= expression*
  - 의미: *variable = variable # (expression)*
  - #는 +, -, *, /, //, % 등의 연산자 중 하나
- 사용예시
  - `var += 3` → `var = var + 3`
  - `t *= 2 + 3` → `t = t * (2 + 3)`

네임스페이스: 변수의 저장공간

- 네임스페이스(name space)[1]: 변수, 함수 등이 정의되어 있는 공간
  - 프로그램이 실행될 때, 전역 네임스페이스(global namespace)를 만듦
  - 새로운 변수가 정의되면, 네임스페이스에 변수의 값을 저장할 공간을 만듦
  - 참고: 함수, 모듈의 내부에서 독립적인 네임스페이스가 만들어짐[2]
- Python tutor example: https://goo.gl/SCbH58

1 명칭 공간으로 번역되기도 함 2 이후 강의에서 다룰 내용

input(): 키보드로부터 입력을 받는 함수

- 표준 입력 (standard input)
  - 프로그램으로 들어가는 데이터 스트림
  - 일반적으로 키보드를 뜻함
- `input()`
  - 인자에 있는 경우, 인자의 내용을 화면에 출력
  - 키보드로 입력받은 내용을 문자열로 반환
    → 변수에 대입하면 그 내용이 변수에 저장됨

int(), float(): 정수 혹은 실수로 형변환

- 형변환(type conversion)
  - 한 자료형(data type)에서 다른 자료형으로 변경하는 것
  - 원하는 연산(예: 사칙연산)을 수행하기 위하여 형변환이 필요
- 형변환 함수: 바뀔형의 이름을 함수처럼 사용
  - `int()`: 인자를 정수형으로 형변환
  - `float()`: 인자를 실수형으로 형변환
- 참고: 형변환이 가능하지 않은 경우에는 오류가 발생함
  - 예: 문자열 'abc'는 정수 혹은 실수로 변환되지 않는다

예제: python에서 키보드 입력

```
name = input('너의 이름은? ')
print('당신은', name, '로군요')

pi = 3.1415926534
# 키보드로부터 값을 입력받음
r = input('Enter a radius: ')
r = float(r) # 연산을 위해 형변환
print(pi * r ** 2)

# input()의 반환값을 float()의 인자로 사용
r = float(input('Enter a radius: '))
print(pi * r ** 2)
```

```
너의 이름은? 미츠하
당신은 미츠하 로군요
Enter a radius: 3
28.2743338806
Enter a radius: 3
28.2743338806
```

참고: 오른쪽 출력결과의 파란색은 키보드 입력을 의미함

제어 흐름 (control flow)

- 명령문(statement)
  - 실행의 단위
  - 통상 한 줄의 코드로 나타나짐
  - 명령문은 한 줄씩 차례로 실행됨
- 제어 흐름은 명령문들이 실행되는 순서/방법을 지시함
  - 조건문: 조건에 따라 다른 명령문들을 실행
  - 반복문: 하나 혹은 그 이상의 명령문들을 반복해서 실행
  - 함수: 동일한 명령문들을 프로그램의 여러 곳에서 실행

조건문과 불리언식

- 조건에 따라 다른 명령문을 실행해야 되는 경우가 있음
- 많은 프로그래밍 언어에서 조건은 불리언식으로 표현됨
  - 불리언식: `True` 혹은 `False`를 계산하는 식(expression)
- 참고: 다음과 같은 값들은 False로 인식됨[1]
  - false로 정의된 상수: `None`, `False`
  - 수를 나타내는 자료형의 0: `0`, `0.0`, `0j`, `Decimal(0)`, `Fraction(0, 1)`
  - 원소를 저장하고 있지 않은 자료구조: `''`, `()`, `[]`, `{}`, `set()`, `range(0)`
  - (이 중 많은 부분은, 앞으로 다룰 내용)

1 https://docs.python.org/3/library/stdtypes.html?highlight=boolean#truth-value-testing

조건문: if, else, elif

```
if condition1:
    statement1_1
    statement1_2
elif condition2:
    statement2_1
    statement2_2
    statement2_3
elif condition3:
    statement3
else:
    else_statement
```

- if: 조건이 참일 때, 코드블럭(code block)을 실행
- elif
  - 추가적인 조건과 그 조건이 참인 경우 실행될 코드블럭을 지정
  - 0개 이상 사용 가능 (즉, 생략 가능하고, 개수의 제한이 없음)
- else
  - 만족하는 조건이 없을 때, 실행되는 코드블럭을 표시
  - 생략되거나 하나의 else절만 사용가능
- 의미
  - 위에서부터 조건을 검사하여 가장 먼처 참이되는 조건에 해당하는 코드블럭만을 실행
  - 만족하는 조건이 없을 때는 else 이후의 코드블럭을 실행

* ␣는 빈칸을 의미. python에서는 통상 4개의 빈칸을 사용하여 code block을 표시함

코드블럭 (code block)

- 코드블럭: 하나의 단위로 실행되는 하나 이상의 명령문들
- python에서는 빈 칸으로 코드블럭을 표시
  - 탭 문자 혹은 빈칸(space)들로 표시
  - 4개의 빈 칸을 사용하는 것을 강력하게 권장함
  - 빈 줄은 무시됨
- 예시

```
if condition:
    statement1
    statement2
```

비교 연산자

연산자	설명	예	결과
<	Less than	5 < 3 3 < 5	False True
>	Greater than	5 > 3 3 > 5	True False
<=	Less than or equal to	3 <= 6	True
>=	Greater than or equal to	4 >= 3	True
==	Equal to	2 == 2 'str' == 'stR' 'str' == 'str'	True False True
!=	Not equal to	'str' != 'stR'	True

불리언 연산자: and, or, not

x	y	x and y	x or y	not x
False	False	False	False	True
False	True	False	True	
True	False	False	True	False
True	True	True	True	

- and: 첫 번째 인자가 True인 경우에만, 두 번째 인자를 계산함 (evaluate)
- or: 첫 번째 인자가 False인 경우만, 두 번째 인자를 계산함 (evaluate)
- not: not 연산자는 불리언 연산자 이외의 연산자보다 우선 순위가 낮음
  - `not a == b`는 `not (a == b)`의 순서로 계산됨
  - `a == not b`는 문법오류를 발생시킴

예제: 조건문 사용 1

```
if 3 < 5:
    print('3 is less than 5')

if 5 < 3:
    print('5 is less than 3')

n = 3
if n < 5:
    print('n is less than 5')

n = 10
if n < 5:
    print('n is less than 5')
```

```
3 is less than 5
n is less than 5
```

예제: 조건문 사용 2 (else절)

```
if 3 < 5:
    print('3 is less than 5')
else:
    print('5 is less than 3')

n = 3
if n < 5:
    print('n is less than 5')
else:
    print('n is NOT less than 5')
```

```
3 is less than 5
n is less than 5
```

예제: 조건문 사용 3 (elif & else절)

```
number = 5
if number < 0:
    print('A negative number')
elif number == 0:
    print('Zero')
elif number < 10:
    print('A single digit number')
else:
    print('A number with two or more digits')
```

```
A single digit number
```

예제: 조건문 사용 4

```
number = 5
if number < 0:
    print('number is a negative number.')
    print('That means number is less than 0.')

if number < 0:
    print('number is a negative number.')
print('It means number is less than 0.')
```

```
It means number is less than 0.
```

예제: 조건문 사용 5 (중첩된 if문)

```
number = 5
if number < 10:
    if number % 2 == 0:
        print('A single digit even number')
    else:
        print('A single digit odd number')
else:
    if number % 2 == 0:
        print('An even number with two or more digits')
    else:
        print('An odd number with two or more digits')
```

```
A single digit odd number.
```

주석 (comment)

- 주석의 필요성
  - 코드에 대한 설명 (다른 사람 & 스스로에게)
- 주석을 다른 방법
  - #를 이용하면, #부터 그 줄의 끝까지는 주석으로 처리됨
  - 여러줄의 주석을 다는 경우
    - 모든 줄에 #를 표시
    - 따옴표 3개를 사용하여, 여러 줄에 걸친 문자열을 사용

Python에서 모듈/패키지 소개

- Python에는 수천~수만 종류의 패키지가 존재함
  - 이미 만들어진 모듈/패캐지를 사용하여 간단하게 프로그램을 작성할 수 있음

- 모듈 (module): 재사용 하고자 하는 변수나 함수의 정의문들을 파일로 저장하여, 다른 프로그램이나 쉘 환경에서 호출하여 사용할 수 있는 방법

- 패키지 (package): 보통 계층 구조를 가지는 여러 모듈들을 모아놓은 것
  - 전체 리스트: https://pypi.python.org/pypi/
  - Python Standard Library: https://docs.python.org/3/library/
  - 유용한 패키지 리스트: https://wiki.python.org/moin/UsefulModules

모듈/패키지 사용하기

- 모듈을 읽어들이는 방법
  - `import module_name`
  - `module_name`은 .py을 뺀 파일이름
- 모듈 사용방법: `module_name.xxx` 와 같이 사용하면 모듈에 속한 함수 혹은 변수 이름을 뜻한다. 사용 예:
  - `module_name.function_name()`
  - `module_name.variable_name`
- 참고: 모듈을 작성, 사용하는 방법은 다시 다룰 예정

예제: math

```
# using math
import math
# constants
print(math.pi)
print(math.e)
# power and logarithmic functions
x = 1.0
print(math.exp(x))   # return e ** x
print(math.log(x))   # return natural log(x)
print(math.log10(x)) # return log(x) with base 10
print(math.sqrt(x))  # return square root(x)
# triangular functions
print(math.cos(x))   # return cos(x)
print(math.sin(x))   # return sin(x)
```

```
3.141592653589793
2.718281828459045
2.718281828459045
0.0
0.0
1.0
0.5403023058681398
0.8414709848078965
```

예제: random

```
import random # using random module
print(random.random())       # float, 0.0 <= x < 1.0
print(random.uniform(1, 10)) # float, 1.0 <= x < 10.0
print(random.randrange(10))  # int, 0 <= x < 10
print(random.randint(1, 10)) # int, 1 <= x <= 10
# choice(): select one element in a given sequence
print(random.choice('abc'))
```

```
0.3677067585658439
1.0474826989553674
0
1
1024
b
```

읽을 거리

- Truth Value Testing: https://docs.python.org/3/library/stdtypes.html?highlight=boolean#truth-value-testing
- `math`: https://docs.python.org/3/library/math.html
- `random`: https://docs.python.org/3/library/random.html

ANY QUESTIONS?