Python 프로그래밍

#2. 파이썬 기초2

목표

- 조건문 (Conditional Statement)
- 반복문 (Loop Statement)
- 함수 (Function)
- List Comprehension

참과 거짓

- 참과 거짓을 표현하는 자료형으로 `True, False`
- 값들 간의 논리연산이나 수치 간의 비교연산의 결과
- `bool()`로 판단
- 비교연산의 종류: `> < == != >= <=`
- 논리연산의 종류: `and or not`

참과 거짓

- t=True, f=False 라고 할때
- 논리연산 and: 둘다 참일때 참
  - (t and t) = True
  - (t and f) = False

참과 거짓

- 논리연산 `or`: 둘 중 하나만 참일때 참
  - `(t or f) = True`
  - `(f or f) = False`
- 논리연산 `not`: 논리 상태를 반전
  - `(not t) = False`
  - `(not f) = True`

조건문 if

- 조건을 평가해 그 결과에 따라 수행 여부를 결정
- 조건문에는 참과 거짓을 판단할 수 있는 문장이 들어간다
- 조건이 참이면 실행 그렇지 않으면 넘어간다

```
if <조건문>:
    <수행할 문장>
```

들여쓰기 (Indentation)

- 파이썬에서는 if <조건문>: 바로 다음 문장부터 수행할 문장이 끝날때까지 들여쓰기를 해야한다
- ':'가 나오면 들여쓰기를 할 것이라 예측하고 보통 스페이스4칸 혹은 2칸으로 정한다

```
if True:
    print("This is true")
    print("welcome!")
```

조건문 if, elif, else

- if조건이 아닐경우 elif를 검사하고 이마저도 아닐경우 else문장

```
if a > 10:
   print("a가 10보다 크다")
elif a == 10:
   print("a가 10이다")
else:
   print("a가 10보다는 작다")
```

단축 평가 (Short-circuit evaluation)

```
a = 0  # or a = '0'
if a and 10 / a:
  print("a가 0입니다.")
else:
  print("에러 없이 통과!")
```

단축 평가 (Short-circuit evaluation)

- 0으로 나누기는 ZeroDivisionError에러를 낸다.
- 하지만 전의 코드는 에러 없이 통과 된다.
- 조건식 전체를 판단하지 않고 수식이 자명하면 더이상 수식을 평가하지 않는다.
  - and 연산에서 좌변이 False이면 우변 피연산자를 평가하지 않는다.
  - or 연산에서는 좌변이 True이면 우변 피연산자를 평가하지 않는다.
  - **파이썬은 항상 좌변이 먼저 평가된다.**

실습

만약 money가 100보다 크면 item = "apple"을, 그렇지 않으면 item = "banana"를 수행하는 코드를 만들어보세요

```
# 처음 money는 10으로 두었을 때
money = 10
if ...
```

반복문 for

- 반복해서 요소들을 처리해야 할 경우
- 시퀀스 데이터나 반복가능한 객체(iterable)에서 원소를 하나하나 뽑아서 변수로 가져온다
- 원소 갯수만큼 반복한다

```
for <변수> in <반복가능한 객체들(리스트, 튜플, 문자열..)>:
    <수행할 문장>
```

반복문 for

결과를 확인해 보세요

```
for i in range(10):
    print(i)
```

range

- range(end): 0부터 end전까지 반복
- range(start, end): start부터 end까지 반복
- range(start, end, step): start부터 end까지 반복하며 step만큼 건너 뜀

반복문 break, continue

- break
  - 반복문을 중단시킨다
- continue
  - 현재 진행하던 반복문 아래의 코드를 진행시키지 않고 반복문의 맨 처음으로 돌아간다

반복문 break, continue

```
for i in range(10):
    if i == 5:
        break
    else:
        print(i)
        continue
```

실습

- 1부터 50까지 출력해보세요
- 구구단을 출력해 보세요

```
-- 1단 --
1 * 1 = 1
1 * 2 = 2
...
9 * 9 = 81
```

- 리스트를 하나 만들고 100이하의 짝수들을 넣어보세요
- 만든 리스트의 모든 원소들을 2로 나눠보세요

리스트의 반복

- 리스트를 반복자로 활용가능하다

```
interest_stocks = ["Naver", "Samsung", "SK Hynix"]
for stock_item in interest_stocks:
  print("buy {}".format(stock_item))
```

딕셔너리의 반복

- 딕셔너리도 반복자로 활용가능하다

```
interest_stocks = {"Naver":10, "Samsung":5, "SK Hynix":30}
for company in interest_stocks.keys():
  print("%s: Buy %s" % (company, interest_stocks[company]))
```

반복문 while

- 반복해서 문장을 수행해야 할 경우
- while <조건문> 에서 조건문이 False가 될때까지 수행한다
- 무한히 실행하는 코드를 짜게될 수 있으니 주의

```
while <조건문>:
    <수행할 문장>
```

반복문 while

다음 코드와 똑같이 동작하는 코드를 while로 만들어 보세요

```
my_list = []
for i in range(100):
    if i % 2 == 0:
        my_list.append(i)
```

함수

- 입력값을 가지고 일을 수행한 다음 결과를 내놓는 것
- 입력값 --> 함수 --> 출력값
- 수학: y = 2x + 1

```
def function(x):
    return 2*x + 1
```

함수

- 여러개의 문장을 하나로 묶어준다
- 묶어둔 문장을 다시 사용할 수 있다
  - 같은 코드를 재사용 할 수 있다.
- 이미 내장되어 있는 함수들도 있다.
  - `type()`, `len()` ...

함수

- 함수 선언은 def로 시작해 콜론(:)으로 끝난다
  - def 키워드로 함수 객체를 만든다.
- 함수의 시작과 끝은 코드의 들여쓰기로 구분한다

```
def <func_name>(<parameter1>, <2>, ... , <N>):
  < statements >
  return < value >
```

함수: return

- 함수를 종료하고 해당 함수를 호출한 곳으로 되돌아가게 한다.
- 반환값으로 어떤 객체도 돌려줄 수 있으며, 반환값이 없을 수도 있다.
- 하나의 객체만 돌려줄 수 있다.

함수: pass

- 아무런 일을 하지 않는다
- 아무것도 하지 않는 함수, 모듈, 클래스를 만들 때 쓴다

```
def function():
    pass
```

스코프 (scope)

- 다음 a 변수는 바뀔까요 안바뀔까요?

```
a = [1, 2, 3]
def func():
    a = [4, 5, 6]
func()
```

스코프 (scope)

- 함수 내부의 이름공간(namespace)을 지역 영역 (local scope)
- 함수 밖의 이름공간을 전역 영역 (global scope)
- 파이썬 자체에서 정의한 내용을 내장 영역 (built-in scope)
- 지역 -> 전역 -> 내장 영역 순으로 찾는다
  - **LGB Rule 이라고 한다**

스코프 (scope)

만약 함수에서 전역 영역의 변수를 쓰고 싶다면: global

```
a = [1, 2, 3]
def func():
    global a
    a = [4, 5, 6]
func()
```

함수: 파라미터 전달 방식

- immutable vs mutable
- mutable 변수가 전달되어 값이 변경되면 함수 외부 값도 변경
- immutable 변수가 전달되고 값이 변경되도 함수 외부 값은 변경 X

함수: 파라미터 전달 방식

- immutable: int, float, bool, str, tuple, range
- mutable: list, dict, set, 유저가 정의한 클래스의 인스턴스

함수: 기본 파라미터

기본 값을 지정할 수 있다

```
def calc(x, y, factor=1):
    return x * y * factor
```

함수: 기본 파라미터

기본 값이 없는 파라미터가 먼저 나와야 한다

```
def calc(factor=1, x, y): # 에러 코드
    return x * y * factor
```

함수: 키워드 인자

인자를 전달할 때 이름을 통해서 전달할 수 있다

```
def report(name, score):
    print(name, score)
report(name="wonkyun", score=80)
```

함수: 가변길이 파라미터

- 파라미터 갯수를 여러개 전달 할 수 있다
- '*' 키워드는 인자의 갯수를 정해두지 않겠다는 뜻

```
def all_sum(*numbers):  # 튜플 형태로 packing.
    total = 0
    for n in numbers:
        total += n
    print(total)
all_sum(1, 2, 3, 4)
```

함수: 가변길이 파라미터

- unpacking: 튜플을 풀어준다

```
t = (1, 2, 3, 4)
all_sum(*t)  # 튜플 형태를 unpacking
```

함수: 정의되지 않은 인자 받기

- '**' 로 딕셔너리로 묶는다

```
def url_builder(domain, **kwargs):
    url = "{}/?".format(domain)
    for key, value in kwargs.items():
        url += "{key}={value}&".format(
            key=key, value=value
        )
    return url
```

재귀함수

- 자기 자신을 호출하는 함수

```
def factorial(i):
    if 1 >= i:
        return 1
    else:
        return i * factorial(i-1)
```

실습: 피보나치 수열

- 1, 1, 2, 3, 5, 8, 13 ...
- 앞에 수를 더한 값이 다음 값으로 온다
- Fi(n) = Fi(n-1) + Fi(n-2)
- 단 Fi(1) = 1, Fi(2) = 1.
- k번째 피보나치 수를 구하는 함수를 만들어 보세요.

익명 함수: Lambda

- 이름이 없는 함수
- `increment_lambda = lambda x: x+1`
- `(lambda x: x+1)(1)`
- Python lambda 의 경우에도 함수형 프로그래밍 언어들에서 제공하는 Higher-Order Function (고차 함수) 들을 제공하고 있다. (Lambda Operator)

익명 함수: Lambda Operator

- map
- filter
- reduce

List Comprehension

- 리스트 컴프리핸션
- 간단한 for문을 만들때 사용한다
- 리스트 객체를 이용해서 조합, 필터링 등 추가 연산을 통해 새로운 리스트를 만든다
- [<표현식> for <아이템> in <리스트> if <조건식>]

실습

- `fruits = ["apple", "banana", "orange", "kiwi"]` 에서 문자열 길이가 5를 초과하는 아이템만 출력하세요
- `fruits`의 아이템을 모두 대문자화 시켜서 출력하세요
- 1부터 100까지 짝수만 제곱시켜서 출력하세요

실습1: 배수의 체크

- 숫자 하나를 받아서 3의 배수이면 "3의 배수 입니다", 5의 배수이면 "5의 배수 입니다", 15의 배수이면 "15의 배수 입니다"를 출력하는 함수를 만들어보세요

```
def check_number(num):
    ... < 함수 내부 > ...


check_number(10)
>>> 5의 배수 입니다.
```

실습2: Palindrome

- "level"같이 앞으로 읽어도 뒤로 읽어도 같은 문자열이 회문이다.
- 회문이면 True, 아니면 False를 리턴하는 함수를 만들어보세요

```
def is_palindrome(string):
    ... < 함수 내부 > ...

is_palindrome("level")
>>> True
```