Basic Python: Part II

Date	@2022년 8월 11일 오전 9:30
Code: Lecture	C2-004_Lecture_02.ipynb
Code: Practice	C2-004_Practice_02.ipynb
Code: Practice-Sol	C2-004_Practice_02-Sol.ipynb
Lecture Note	C2-004_Lecture 02-basic 2.pdf
No.	2

▼ 1. 제어

▼ 1-1) 조건문: if

- **if문의 기본 구조**

```
if 조건문:
  수행할 문장1
  수행할 문장2
  ...
else:
  수행할 문장A
  수행할 문장B
...
```

: 조건문을 확인하여 참이면 if문 바로 다음 문장(if 블록)들을 수행하고, 조건문이 거짓이면 else문 다음 문장(else 블록)들을 수행하게 된다. 따라서 else문은 if문 없이 사용할 수 없다.

- **if문 사용 예시**

```
# 조건문을 만족하는 경우
weather = "비"

if weather == "비":
  print("우산을 챙기세요") # 출력 O
```

```
# 조건문을 만족하지 않는 경우
weather = "해"

if weather == "해":
  print("우산을 챙기세요") # 출력 X
```

```
# 다양한 조건을 판단하는 elif
weather = "해"
# weather = input("오늘 날씨는 어때요?")

if weather == "비" or weather == "눈":
  print("우산을 챙기세요.")
elif weather == "해":
  print("선글라스를 챙기세요.")
else:
  print("준비물은 없습니다.")
```

▼ 1-2) 반복문: for

- **for문의 기본 구조**

```
for 변수 in 리스트(또는 튜플, 문자열):
  수행할 문장1
  수행할 문장2
```

: 리스타나 튜플, 문자열의 첫 번째 요소부터 마지막 요소까지 차례대로 변수에 대입되어 “수행할 문장1”, “수행할 문장2” 등이 수행된다.

- **for문 사용 예시**

```
# for문이 없다면
print("대기번호: 0")
print("대기번호: 1")
print("대기번호: 2")
print("대기번호: 3")
print("대기번호: 4")
```

```
# for문 활용
for waiting_num in [0, 1, 2, 3, 4]:
  print("대기번호: {0}".format(waiting_num))
```

```
# for문 + range() 활용
for waiting_num in range(5): # 0, 1, 2, 3, 4
  print("대기번호: {0}".format(waiting_num))

print("\n")

# for문 + range() 활용
for waiting_num in range(1, 5): # 1, 2, 3, 4
  print("대기번호: {0}".format(waiting_num))
```

```
cafe = ["A-110", "B-272", "A-111"]

for customer in cafe:
  print("{0}번 손님, 커피가 준비되었습니다.".format(customer))
```

- **for문의 응용**

```
# for문 + if문 활용
marks = [90, 25, 67, 45, 80]

num_student = 0
for score in marks:
  num_student += 1

  if score >= 60:
    print("%d번 학생은 합격입니다." % num_student)
  else:
    print("%d번 학생은 불합격입니다." % num_student)
```

```
# for문 + if문 + range() 활용
marks = [90, 25, 67, 45, 80]

num_student = 0
for num_student in range(len(marks)):

  score = marks[num_student]
  if score >= 60:
    print("%d번 학생은 합격입니다." % num_student)
  else:
    print("%d번 학생은 불합격입니다." % num_student)
```

```
# 이중 for문 + range() 활용: 구구단1
for i in range(2, 10):
  for j in range(1, 10):
    print(i*j)
  print("\n")
```

```
# 이중 for문 + range() 활용: 구구단2
for i in range(2, 10):
  print("<구구단: {0}단>".format(i))
  for j in range(1, 10):
    print("{0} x {1} = {2}".format(i, j, i*j))
  print("\n")
```

- **한 줄 for문**

```
# 출석번호를 1의 자리에서 100의 자리 수로 변경
students = [1, 2, 3, 4, 5]
print(students)

new_students = [100+i for i in students]
print(new_students)
```

```
# 학생 이름을 대문자로 변환
students = ["iron man", "thor", "spider man"]

new_students = [i.upper() for i in students]
print(new_students)
```

▼ 1-3) 반복문: while

- **while문의 기본 구조**

```
while 조건문:
  수행할 문장1
  수행할 문장2
  ...
```

: 조건문이 참인 동안에 while문 아래의 “수행할 문장”이 반복해서 수행된다.

- **while문 사용 예시**

```
# while문 사용 예제1
hit = 0
while hit < 10:
  hit += 1
  print("나무를 %d번 찍었습니다." % hit)

  if hit == 10:
    print("나무가 넘어갔습니다.")
```

```
# while문 사용 예제2
customer = "김학구"
person = "Unknown"

while person != customer:
  print("{0} 손님, 커피가 준비되었습니다.".format(customer))
  person = input("성함이 어떻게 되세요? ")
```

▼ 1-4) continue와 break

- **continue문:** while문을 빠져나가지 않고, while문의 맨 처음(조건문)으로 돌아가게 하는 명령어

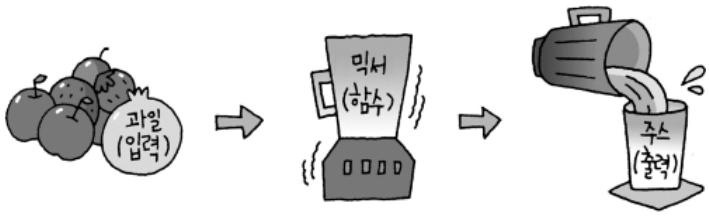
```
# continue 사용 예제
absent = [2, 5] # 결석한 학생
for student in range(1, 11):
  if student in absent:
    continue
  print("{0}번 학생, 과제 제출하세요.".format(student))
```

- **break문:** while문을 강제로 멈추고 빠져나가게 하는 명령어

```
# break 사용 예제
absent = [2, 5] # 결석한 학생
no_homework = [7] # 과제를 안 한 학생
for student in range(1, 11):
  if student in absent:
    continue
  elif student in no_homework:
    print("오늘 수업은 여기에서 마치겠습니다. {0}번 학생은 교무실로 오세요.".format(student))
    break
  print("{0}번 학생, 과제 제출하세요.".format(student))
```

▼ 2. 함수

▼ 2-1) 함수



- 함수가 하는 일은 위 그림처럼 입력값을 가지고 어떤 일을 수행한 다음에 그 결과물을 내어놓는 것 [Jump to Python]

- **함수를 사용하는 이유**
  - 똑같은 내용을 반복해서 작성해야할 때, 반복적으로 사용되는 가치있는 부분을 “하나의 뭉치로 묶어서 어떤 입력값을 주었을 때 어떤 결과값을 돌려준다”라는 식의 함수로 작성하는 것이 현명
  - 여러 기능들을 각각 함수화하면 프로그램의 흐름을 일목요연하게 확인할 수 있음

- **함수의 구조**

```
def 함수명(매개변수):
  수행할 문장1
  수행할 문장2
  ...
```

- **함수 사용의 예시**

```
# 함수 정의
def open_account():
  print("새로운 계좌가 생성되었습니다.")
```

```
# 함수 실행
def open_account():
  print("새로운 계좌가 생성되었습니다.")

open_account()
```

▼ 2-2) 전달값과 반환값

```
# 전달값과 반환값을 가지는 함수 예제1
def deposit(balance, money): # (전달값) balance: 현재 잔액, money: 입금 금액
  print("입금이 완료되었습니다. 잔액은 {0}원 입니다.".format(balance + money))
  return balance + money # (반환값) balance + money

balance = 0
balance = deposit(balance, 1000)

print(balance)
```

```
# 전달값과 반환값을 가지는 함수 예제2
def deposit(balance, money): # 입금
  print("입금이 완료되었습니다. 잔액은 {0}원 입니다.".format(balance + money))
  return balance + money

def withdraw(balance, money): # 출금
  if balance >= money:
    print("출금이 완료되었습니다. 잔액은 {0}원 입니다.".format(balance - money))
    return balance - money
  else:
    print("잔액이 부족합니다. 출금 가능한 최대금액은 {0}원 입니다.".format(balance))
    return balance

balance = 0
balance = deposit(balance, 2000)  # 2,000원 입금 (출금 후 잔액: 2,000원)
balance = withdraw(balance, 500)  #   500원 출금 (출금 후 잔액: 1,500원)
balance = withdraw(balance, 1700) # 1,700원 출금 (출금 전 잔액: 1,500원)
```

```
# 전달값과 반환값을 가지는 함수 예제3
def deposit(balance, money): # 입금
  print("입금이 완료되었습니다. 잔액은 {0}원 입니다.".format(balance + money))
  return balance + money

def withdraw_night(balance, money): # 수수료 있는 출금
  commission = 200 # 수수료 200원
  return commission, balance - money - commission

balance = 0
balance = deposit(balance, 2000)
commission, balance = withdraw_night(balance, 500)
print("수수료는 {0}원이며, 잔액은 {1}원 입니다.".format(commission, balance))
```

▼ 2-3) 기본값

```
# 함수의 전달값이 다양할 경우
def profile(name, age, main_lang):
  print("이름: {0}\t나이: {1}살\t주 사용 언어: {2}".format(name, age, main_lang))

profile("아이언맨", 41, "Python")
profile("토르", 42, "Java")
```

```
# 함수의 전달값 중 기본적으로 사용하는 값이 있을 경우
def profile(name, age=20, main="격투"):
  print("이름: {0}\t나이: {1}살\t주특기: {2}".format(name, age, main))

profile("손오공")
profile("배지터")
profile("피카츄", 5, "전기공격")
```

▼ 2-4) 키워드값

```
# 키워드값을 이용한 함수 호출 예제
def profile(name, age, main_lang):
  print("이름: {0}\t나이: {1}살\t주 사용 언어: {2}".format(name, age, main_lang))

profile(name="아이언맨", main_lang="Python", age=41)
profile(age=42, main_lang="Java", name="토르")
```

▼ 2-5) 가변인자

```
# 가변인자를 이용하지 않은 함수 예제
def profile(name, age, lang1, lang2, lang3, lang4, lang5):
  print("이름: {0}\t나이: {1}살\t".format(name, age), end=" ")
  print(lang1, lang2, lang3, lang4, lang5)

profile("아이언맨", 41, "Python", "Java", "C", "C++", "HTML")
profile("토르", 41, "MATLAB", "Swift", "", "", "")
```

```
# 가변인자를 이용한 함수 예제
def profile(name, age, *languages):
  print("이름: {0}\t나이: {1}살\t".format(name, age), end=" ")
  for lang in languages:
    print(lang, end=" ")
  print()

profile("아이언맨", 41, "Python", "Java", "C", "C++", "HTML", "C#")
profile("토르", 41, "MATLAB", "Swift")
```

▼ 2-6) 지역변수와 전역변수

- **지역변수:** 함수 내에서만 사용할 수 있는 변수 (함수호출이 끝나면 사라짐)
- **전역변수:** 프로그램 내 어디서든지 사용할 수 있는 변수

```
# 지역변수 예제
gun = 10

def checkpoint(soldiers): # 경계근무
  # gun = 20
  gun = gun - soldiers  # Error: 함수 내에 gun 이라는 변수가 정의되어 있지 않음
  print("[함수 내] 남은 총: {0}".format(gun))

print("전체 총: {0}".format(gun))
checkpoint(2) # 2명이 경계 근무 나감
print("남은 총: {0}".format(gun))
```

```
# 전역변수 예제
gun = 10

def checkpoint(soldiers): # 경계근무
  global gun # 전역 공간에 있는 gun 사용
```

```
  gun = gun - soldiers
  print("[함수 내] 남은 총: {0}".format(gun))

print("전체 총: {0}".format(gun))
checkpoint(2) # 2명이 경계 근무 나감
print("남은 총: {0}".format(gun))
```

: 일반적으로 전역변수는 많이 사용하게 되면 코드 관리가 어려워지므로 권장되지 않음

```
# 전달값을 이용하여 전역변수 사용을 피하는 예제
gun = 10

def checkpoint_return(gun, soldiers):
  gun = gun - soldiers
  print("[함수 내] 남은 총: {0}".format(gun))
  return gun

print("전체 총: {0}".format(gun))
gun = checkpoint_return(gun, 2) # 2명이 경계 근무 나감
print("남은 총: {0}".format(gun))
```

▼ 3. 입출력

▼ 3-1) 표준입출력

```
print("Python", "Java")  # Python Java
print("Python" + "Java") # PythonJava

print("Python", "Java", sep=",") # Python,Java
print("Python", "Java", sep=" vs ") # Python vs Java

print("Python", "Java", sep=",", end="?")
print("무엇이 더 재미있을까요?")
```

```
import sys
print("Python", "Java", file=sys.stdout) # 표준출력으로 처리
print("Python", "Java", file=sys.stderr) # 표준에러로 처리
```

```
# 정렬 예제: 시험점수
scores = {"수학": 90, "영어": 0, "코딩": 100}
for subject, score in scores.items():
  print(subject, score)
print()

scores = {"수학": 90, "영어": 0, "코딩": 100}
for subject, score in scores.items():
  print(subject.ljust(6), str(score).rjust(4), sep=":")
```

```
# 정렬 예제: 은행 대기순번표
# 001, 002, 003, ...
for num in range(1, 21):
  print("대기번호: " + str(num))
print()

for num in range(1, 21):
  print("대기번호: " + str(num).zfill(3))
```

```
# 표준 입력
answer = input("값을 입력하세요: ")
print("입력하신 값은 {0} 입니다.".format(answer))
```

▼ 3-2) 다양한 출력포맷

```
# 빈 자리는 빈 공간으로 두고, 오른쪽 정렬을 하되, 총 10자리 공간을 확보
print("{0: >10}".format(500))
```

```
# 빈 자리는 빈 공간으로 두고, 오른쪽 정렬을 하되, 총 10자리 공간을 확보
# 양수일 때는 +, 음수일 때는 -로 표시
print("{0: >+10}".format(500))
print("{0: >+10}".format(-500))
```

```
# 빈 자리는 _로 채우고, 왼쪽 정렬을 하되, 총 10자리 공간을 확보
# 양수일 때는 +, 음수일 때는 -로 표시
print("{0:_<+10}".format(500))
```

```
# 세 자리수마다 콤마 표시
print("{0:,}".format(129823456))
```

```
# 세 자리수마다 콤마 표시
# 양수일 때는 +, 음수일 때는 -로 표시
print("{0:+,}".format(129823456))
print("{0:+,}".format(-129823456))
```

```
# 빈 자리는 ^로 채우고, 왼쪽 정렬을 하되, 총 30자리 공간을 확보
# 세 자리수마다 콤마 표시
# 양수일 때는 +, 음수일 때는 -로 표시
print("{0:^<+20,}".format(129823456))
print("{0:^<+20,}".format(-129823456))
```

```
# 소수점 출력
print("{0:f}".format(5/3))

# 소수점 특정 자리수까지만 표시
print("{0:.2f}".format(5/3)) # 소수점 3째 자리에서 반올림하여 소수점 2째 자리까지만 표시
```

▼ 3-3) 파일입출력

```
# 파일 쓰기
score_file = open("score.txt", "w", encoding="utf8")

print("수학: 80", file=score_file)
print("영어: 0", file=score_file)

score_file.close()
```

```
# 파일 덮어쓰기
score_file = open("score.txt", "a", encoding="utf8")

score_file.write("과학: 90")
score_file.write("\n코딩: 100")

score_file.close()
```

```
# 파일 읽기 - 파일 내용 한 번에 불러오기
score_file = open("score.txt", "r", encoding="utf8")

print(score_file.read())

score_file.close()
```

```
# 파일 읽기 - 파일 내용 한 줄씩 불러오기
score_file = open("score.txt", "r", encoding="utf8")

print(score_file.readline()) # 한 줄 읽고, 커서는 다음 줄로 이동
print(score_file.readline())
print(score_file.readline())
print(score_file.readline())

score_file.close()
```

```
# 파일 읽기 - 임의의 파일 내용 불러오기
score_file = open("score.txt", "r", encoding="utf8")

while True:
  line = score_file.readline()
  if not line:
    break
  print(line, end="")

score_file.close()
```

```
# 파일 읽기 - 모든 내용을 가져와서 리스트 형태로 저장
score_file = open("score.txt", "r", encoding="utf8")

lines = score_file.readlines()
for line in lines:
  print(line, end="")

score_file.close()
```

▼ 3-4) pickle

- 프로그램 상에서 우리가 사용하고 있는 데이터를 파일 형태로 저장하고 읽어올 수 있게 해주는 모듈

```
# pickle 사용하여 파일 쓰기
import pickle

profile_file = open("profile.pickle", "wb") # wb: binary type 쓰기

profile = {"이름":"손오공", "나이":20, "취미":["축구", "골프", "코딩"]}
print(profile)
pickle.dump(profile, profile_file) # profile에 있는 정보를 파일에 저장

profile_file.close()
```

```
# pickle 사용하여 파일 읽기
import pickle

profile_file = open("profile.pickle", "rb") # wb: binary type 읽기

profile = pickle.load(profile_file) # 파일에 있는 정보를 profile에 불러오기
print(profile)

profile_file.close()
```

▼ 3-5) with

- 파일입출력을 보다 간편하게 수행할 수 있는 명령어

```
import pickle

with open("profile.pickle", "rb") as profile_file:
  print(pickle.load(profile_file))
```


```
import pickle

with open("profile.pickle", "rb") as profile_file:
  print(pickle.load(profile_file))
```