Python Programming

For Beginner

File

For Python 3

파일 기본

1. 파일 객체
   - 파일의 내용을 읽는 수단
   - 테이프 드라이브나 DVD 재생기
   - 파일 포인터가 읽는 위치를 가리킨다 (마치 레코드 플레이어 헤드처럼).
2. 파일 객체 열기
   - open(file, mode= ' r ' , buffering=-1, encoding=**None**, errors=**None**, newline=**None**, closefd=**True**, opener=**None**)
   - pydoc open
3. Open함수의 인자
   - File → 필수 값, 파일 경로
   - Mode → r, w, x, a, b, t, +, U
   - Buffering → 0, 1, 1보다 큰 양수
   - Encoding → utf8 명시 (python의 인코딩 목록)
4. 파일 객체 닫기
   - Close()
   - 닫지 않으면 버퍼링 된 데이터 소실

파일 읽기

1. read()
   - 파일의 전체 내용 반환
2. read(글자수)
   - 글자 수만큼 읽고 파일 포인터를 끝으로 이동
   - 파일의 끝까지 읽은 후 빈 문자 반환
3. seek(포인터 위치)
   - 포인터를 특정 위치로 이동
   - 맨 앞으로 이동은 seek(0)
4. readline()
   - 파일에서 한 줄 씩 읽기
5. readlines()
   - 파일에서 한 줄 씩 읽고 리스트 형식으로 반환

```
#실행 인자와 입력 함수 조합
from sys import argv

script, filename = argv

txt = open(filename)
print(f"선택한 파일: {filename}")
print(txt.read())

print("파일 이름을 다시 한 번 입력하세요.")
file_again = input("> ")

txt_again = open(file_again)
print(txt_again.read())
```

파일 쓰기

1. 파일을 쓰기로 오픈
   - Mode = wt (쓰기 텍스트 모드)
   - Mode = at (추가 텍스트 모드)
2. Write('내용')
   - 파일에 '내용'을 쓴다.
3. ~~writeline('내용')~~
   - 대신 write(" 내용₩n")
4. Writelines(['내용 1', '내용 2', ...])
   - 리스트 각 원소의 문자열 끝에 반드시 ₩n 사용
5. Truncate()
   - 파일 내용 비우기

```
#파일 쓰기
wf=open('sample_write.txt', 'wt', encoding='utf-8')
help(wf)

wf.write('파이썬으로 파일을 작성하고 있습니다.\n')
wf.write('write()호출 후 반환 숫자는 글자 수 입니다.\n')
wf.writelines(['Writeline()은 없습니다.\n','writelines
()는 리스트 형식을 받습니다.\n'])
wf.close()
```

```
#파일 내용 추가하기
af=open('sample_write.txt', mode='at',
encoding='utf-8')
af.writelines(['--------------------\n',
'파일을 추가 모드로 열었습니다.\n','덧 붙인
내용입니다.\n'])
af.close()
```

Finally와 with 구문으로 파일 다루기

1. Try ~ finally 구문

- 문제가 발생 해도 Finally 절은 실행

```
try:
    변수 = open(파일경로, 옵션)
    … 파일 조작 …
finally:
    변수.close()
```

```python
#try ~ finally 구문
try:
    f = open('finallywith.txt', mode='wt', encoding='utf-8')
    f.write('파이썬으로 파일을 작성하고 있습니다.\n')
    f.write('try~finally문을 사용합니다.\n')
finally:
    f.write('finally 절의 close()는 무조건 실행 됩니다.')
    f.close()
```

2. With 블록

- 명시적 close() 메서드 호출 필요 없음.

```
with open(파일경로) as 변수:
    파일 객체 처리
```

```python
#with 블록
with open('with.txt', mode='wt', encoding='utf-8') as f:
    f.write('파이썬으로 파일을 작성하고 있습니다.\n')
    f.write('with 블록을 사용합니다.\n')
    f.write('close()는 자동으로 호출됩니다')
```

응용 – 파일 복사

1. from os.path import exists
2. len(입력_자료)
   - 문자열 길이를 숫자로 반환
3. open(원본 파일).read()
   - close() 호출 필요 없음
4. 문자열을 따옴표로 제대로 마치지 못한 경우
   - EOL while scanning string literal error

```
from sys import argv
from os.path import exists

script, from_file, to_file = argv
print(f"{from_file}에서 to {to_file}로 복사합니다.")

in_file = open(from_file, encoding='utf-8')
indata = in_file.read()
print(f"입력 파일의 길이는 {len(indata)}바이트 입니다.")

print(f"대상 파일이 존재여부? {exists(to_file)}")
print("계속 하려면 ENTER, 중지하려면 CTRL-C를 입력하세요.")
input()

out_file = open(to_file, 'w', encoding='utf-8')
out_file.write(indata)
print("모든 작업이 끝났습니다.")

out_file.close()
in_file.close()
```

Function

For Python 3

함수 개요

1. 구조적/절차적 프로그래밍 용어
2. 여러 구문을 하나로 묶은 실행 단위
   - 기능의 재사용 촉진
3. Python 함수의 3가지 특징
   - 코드 조각에 이름 부여
   - 실행인자를 받을 수 있다
   - 작은 스크립트나 작은 명령을 만든다.
4. Python 함수의 구조
   - return은 생략 가능
   - 파라미터로 *args 사용해 argv 효과

```
def 함수_이름(파라미터):
    실행 코드
    (옵션)return 결과 값
```

```python
# *args로 스크립트의 argv와 비슷한 동작
def print_two(*args):
    arg1, arg2 = args
    print(f"arg1: {arg1}, arg2: {arg2}")

# *args 대신 이렇게 해도 됩니다.
def print_two_again(arg1, arg2):
    print(f"arg1: {arg1}, arg2: {arg2}")

# 실행 인자 하나만 받는 함수
def print_one(arg1):
    print(f"arg1: {arg1}")

# 실행 인자가 없는 함수
def print_none():
    print("아무런 실행 인자도 받지 않음.")
```

함수 인자

1. 기본 값 인자
   - def function(a, b=value):
   - 인자를 입력하지 않으면 기본 값 적용
2. 위치 인자
   - def function(a, b, c, ...)
   - 함수에서 정의한 대로 적용
3. 키워드 인자
   - 키워드를 사용해 순서 무시 입력
   - 위치 인자와 키워드 인자 혼합 (위치 인자 먼저)

```
#기본 인자
def choose_car(use,type='전기차'):
    print("""
        {use} 용도로,
        {type}를 선택했습니다.
    """.format(use=use,type=type))

choose_car('출퇴근')

#위치 인자
choose_car('캠핑', '디젤차')

#키워드 인자
choose_car(type='가솔린차',use='출장')
choose_car('친환경',type='수소차')
```

함수와 변수

1. 함수에 실행인자를 전달하는 방법
   - 값 그 자체
   - 변수
   - 계산
2. 함수 내의 변수와 스크립트 변수
   - 변수의 생존 범위가 다르다.
   - 함수 밖에서 함수로 전달된 변수는 임시 변수로 동작 후 소멸
   - 함수 변수와 전역 변수는 다른 이름으로
3. 함수 내에서 함수 실행 가능
4. 함수에 전달하는 실행 인자의 수
   - 최대 5개 권장

```
#함수와 변수
def cheese_and_crackers(cheese_count, boxes_of_crackers):
    print(f"치즈가 {cheese_count}개나 있어요!")
    print(f"크래커가 {boxes_of_crackers}상자나 있어요!")
    print("파티를 열기에 충분하네요!")
    print("담요 한 장은 가져와요.\n")
cheese_and_crackers(20, 30)

amount_of_cheese = 10
amount_of_crackers = 50
cheese_and_crackers(amount_of_cheese, amount_of_crackers)

cheese_and_crackers(10 + 20, 5 + 6)

cheese_and_crackers(amount_of_cheese + 100,
amount_of_crackers + 1000)
```

함수와 파일

1. Python에서 모든 타입은 객체
2. 함수에 파일 객체 전달
   - 함수 호출 전 open() 함수 호출
   - 함수 호출 후 close() 함수 호출
3. 전달된 파일 객체의 읽기와 쓰기
   - 파일 객체의 멤버 함수 이용
   - read(), readline(), readlines()
   - write(), writelines()
   - seek()

```
from sys import argv

script, input_file = argv

def print_all(f):
    print(f.read())

def rewind(f):
    f.seek(0)

def print_a_line(line_count, f):
    print(line_count, f.readline())

current_file = open(input_file, encoding='utf-8')
```

반환하는 함수

1. 함수가 반환하는 값을 변수로 받기
2. return 문
   - 결과 값 반환
   - 함수 종료
3. 함수 실행 인자에 다른 함수 반환 값 사용

```python
#함수의 반환
def add(a, b):
    print(f"더하기 {a} + {b}")
    return a + b

def subtract(a, b):
    print(f"빼기 {a} - {b}")
    return a - b

def multiply(a, b):
    print(f"곱하기 {a} * {b}")
    return a * b

def divide(a, b):
    print(f"나누기 {a} / {b}")
    return a / b

def even_or_odd(n):
    if n % 2 == 0:
        print("짝수")
        return
    print("홀수")
```

수고했습니다!