프로그래밍을 할 때 같은 기능을 여러 방법으로 구현할 수 있다. 각 챕터의 연습문제 중 코드 구현과 관련된 문제는 보기의 코드 외의 방식으로도 구현할 수 있다. 추가로 제공된 코드 실행 시 출력되는 결과는 연습문제의 정답과 동일하다.

챕터 3

문제 10) [객관식] 사용자 입력으로 두 int형 숫자를 받아 두 수의 합을 출력하려 한다. 아래의 코드 중 문제의 의도대로 프로그래밍하지 않은 것을 골라 보자(예시 입/출력 참고).

<table>
<tr><th colspan="2">보기</th><th>예시</th></tr>
<tr><td>ⓐ</td><td>a = input()
b = input()
c = int(a) + int(b)
print(c)</td><td rowspan="4">입력1: 3[enter]4[enter]
출력1: 7

입력2: 10[enter]5[enter]
출력2: 15

입력3: -5[enter]4[enter]
출력3: -1</td></tr>
<tr><td>ⓑ</td><td>a = int(input())
b = int(input())
c = a + b
print(c)</td></tr>
<tr><td>ⓒ</td><td>a = input()
b = input()
c = int(a + b)
print(c)</td></tr>
<tr><td>ⓓ</td><td>c = int(input() + input())
print(c)</td></tr>
</table>

해설

ⓓ를 제외하면 모두 입력받은 두 숫자의 합을 구하여 출력한다. 이를 다른 방식으로 구현한 코드들은 아래와 같다.

코드
a, b = int(input()), int(input()) c = a+b
c = int(input()) + int(input()) print(c)
print(int(input()) + int(input()))

문제 11) [객관식] 띄어쓰기로 구분된 단어들을 입력하고 구별된 각 단어들 중 사전 순서 상 가장 먼저 등장하는 단어를 출력하려 한다. 해당 의도대로 프로그래밍하기 위해 빈칸에 올 수 있는 코드를 골라 보자(예시 입/출력 참고).

<table>
<tr><th>문제</th><th>예시</th><th>보기</th></tr>
<tr><td rowspan="4">words = input()
words_list = words.split()
→ (ⓐ / ⓑ / ⓒ / ⓓ)
print(first)</td><td rowspan="4">입력1: fox ape zebra
출력1: ape

입력2: abc def
출력2 abc</td><td>ⓐ first = max(words_list)</td></tr>
<tr><td>ⓑ first = max(words)</td></tr>
<tr><td>ⓒ first = min(words_list)</td></tr>
<tr><td>ⓓ first = min(words)</td></tr>
</table>

해설

ⓒ 문자열의 리스트 중 사전 순서상 가장 먼저 등장하는 단어를 first에 대입시킨다. 이를 다른 방식으로 구현한 코드는 아래와 같다.

<table>
<tr><th>코드</th></tr>
<tr><td>words_list.sort()
first = words_list[0]</td></tr>
<tr><td>first = sorted(words_list)[0]</td></tr>
</table>

챕터 5

문제 5) [객관식] 아래의 코드는 파일 이름 문자열 file에서 확장자가 .jpg인 파일 이름만 출력하는 프로그램의 일부이다. 그런데 소문자 확장자 .jpg뿐만 아니라 대문자가 확장자에 포함된 경우에도 출력하도록(아래의 코드1, 코드2 모두 file이 출력되도록) 설계하고 싶다. 다음 중 의도한 동작을 수행할 수 있는 코드를 골라 보자.

코드1	코드2	보기
file = "cat1.jpg" → if ( ): print(file)	file = "cat2.JPG" → if ( ): print(file)	ⓐ ".jpg" in file ⓑ ".jpg".upper() in file ⓒ ".jpg" in file.upper() ⓓ ".JPG" in file.upper()

해설

ⓓ 문자열 변수 file의 확장자가 대소문자에 관계없이 .jpg인지 확인한다. 이를 다른 방식으로 구현한 코드는 아래와 같다.

코드
".jpg" in file.lower()

챕터 6

문제 6) [객관식] 아래의 코드들은 피보나치 수열을 구하는 알고리즘의 코드이다. 피보나치 수열은 첫 두 숫자는 1, 1로 시작하여 이후의 수열은 직전 두 수의 합으로 나열되는 수열이다. 예를 들어, 피보나치 수열을 7번째 숫자까지 구하면 [1, 1, 2, 3, 5, 8, 13]이다. 아래의 코드 중, 마지막 줄의 print(fibo)에 의해 출력되는 결과가 다른 하나를 골라 보자.

보기	

ⓐ	n = 10
	fibo = [1, 1]
	for i in range(n):
	  num = fibo[-1] + fibo[-2]
	  fibo = fibo + [num]
	print(fibo)

ⓑ	n = 10
	fibo = []
	for i in range(n):
	  if i == 0 or i == 1:
	  fibo = fibo + [1]
	  else:
	  num = fibo[-1] + fibo[-2]
	  fibo = fibo + [num]
	print(fibo)

ⓒ	n = 10
	fibo = [1, 1]
	for i in range(n):
	  if i == 0 or i == 1:
	  continue
	  num = fibo[-1] + fibo[-2]
	  fibo = fibo + [num]
	print(fibo)

ⓓ	n = 10
	fibo = [1, 1]
	while len(fibo) < n:
	  num = sum(fibo[-2:])
	  fibo = fibo + [num]
	print(fibo)

해설

ⓐ를 제외하면 모두 피보나치 수열을 n번째(10번째) 숫자까지 구한다. fibo 리스트를 다른 방식으로 구현한 코드는 아래와 같다.

코드

n = 10
fibo = [1, 1]
for i in range(2, n):
  num = fibo[-1] + fibo[-2]
  fibo = fibo + [num]
print(fibo)

n = 10
fibo = [1, 1]
for i in range(n):
  num = fibo[-1] + fibo[-2]
  if i > 1:
  fibo = fibo + [num]
print(fibo)

n=10 fibo = [1, 1] while len(fibo) < 10:   fibo = fibo + [sum(fibo[-2:])] print(fibo)

챕터 9

문제 6) 이름 문자열과 나이를 각각 키-밸류 쌍으로 구성한 딕셔너리 name_age에 대해서, 아래의 문제들을 해결해 보자.

① name_age의 키-밸류 쌍 중, 나이가 20 이상인 이름들을 adults 리스트에 담은 후 출력하려 한다. 빈칸을 채워 알고리즘을 완성하려 할 때, 빈칸에 들어갈 코드로 적절한 것을 골라 보자.

코드

```
name_age = {"Kim": 10, "Lee": 15, "Park": 20, "Choi": 25}
adults = []
(          )
print(adults)
```

보기

ⓐ
```
for k in name_age:
    if k >= 20:
        adults.append(name_age[k])
```

ⓑ
```
for k, v in name_age:
    if k >= 20:
        adults.append(v)
```

ⓒ
```
for k, v in name_age.items():
    if v >= 20:
        adults.append(name_age[v])
```

ⓓ
```
for k, v in name_age.items():
    if v >= 20:
        adults.append(k)
```

해설

ⓓ name_age 딕셔너리에서 키-밸류 쌍을 확인하여 밸류 값이 20 이상이면 키 값을 adults 리스트에 추가한다. 이를 다른 방식으로 구현한 코드는 아래와 같다.

코드

```
for k in name_age:
    if name_age[k] >= 20:
        adults.append(k)
```

```
for k in name_age.keys():
    if name_age[k] >= 20:
        adults.append(k)
```

```
adults = [k for k, v in name_age.items() if v >= 20]
```

② name_age의 키-밸류 쌍 중, 나이가 20 이상인 이름-나이의 키-밸류 쌍만을 추려서 모은 딕셔너리 adult_name_age를 구하여 출력하려 한다. 빈칸에 들어갈 코드로 적절한 것을 골라 보자.

코드

```
name_age = {"Kim": 10, "Lee": 15, "Park": 20, "Choi": 25}
adult_name_age = dict()
(          )
print(adult_name_age)
```

보기

ⓐ
```
for k in name_age:
    if k >= 20:
        adult_name_age[k] = name_age[k]
```

ⓑ
```
for k in name_age:
    if k >= 20:
        adult_name_age[k] = k
```

ⓒ
```
for k, v in name_age.items():
    if v >= 20:
        adult_name_age[k] = v
```

ⓓ
```
for v in name_age.values():
    if v >= 20:
        adult_name_age[v] = v
```

해설

ⓒ name_age의 각 키-밸류 쌍을 반복하여 밸류 값이 20 이상일 때의 키-밸류 쌍을 adult_name_age 딕셔너리에 추가한 것이다. 이를 다른 방식으로 구현한 코드는 아래와 같다.

코드

```
for k in name_age:
    if name_age[k] >= 20:
        adult_name_age[k] = name_age[k]
```

```
for k in name_age.keys():
    if name_age[k] >= 20:
        adult_name_age[k] = name_age[k]
```

```
adult_name_age = {k: v for k, v in name_age.items() if v >= 20}
```

③ 새로운 딕셔너리 adult_names를 만들어 나이에 따라 리스트에 구분하여 담고자 한다. name_age의 키-밸류 쌍 중, 나이가 20 이상인 이름들을 adult_names의 "adult" 키에 대응하는 밸류 리스트에, 20보다 작은 이름들을 adult_names의 "non-adult" 키에 대응하는 밸류 리스트에 담는다. 아래의 코드에 대해 예제 출력과 같은 결과를 얻고자 할 때, 빈칸에 들어갈 코드로 적절한 것을 골라 보자.

코드

```
name_age = {"Kim": 10, "Lee": 15, "Park": 20, "Choi": 25}
adult_names = {"adult": [], "non-adult": []}
(          )
print(adult_names)
```

실행 결과

```
{'adult': ['Park', 'Choi'], 'non-adult': ['Kim', 'Lee']}
```

보기

ⓐ
```
for k in name_age:
    if name_age[k] >= 20:
        adult_names["adult"].append(k)
```

ⓑ
```
for k in name_age:
    if name_age[k] >= 20:
        adult_names["adult"].append(k)
    else:
        adult_names["non-adult"].append(k)
```

ⓒ
```
for k in name_age:
    if v >= 20:
        adult_names["adult"].append(k)
    else:
        adult_names["non-adult"].append(k)
```

ⓓ
```
for v in name_age:
    if v >= 20:
        adult_names["adult"].append(v)
    else:
        adult_names["non-adult"].append(v)
```

해설

ⓑ name_age의 각 키-밸류 쌍을 반복하여 밸류 값이 20 이상인 경우에는 adult_names["adult"] 리스트에, 그렇지 않은 경우 adult_names["non-adult"] 리스트에 키 값을 추가한다. 이를 다른 방식으로 구현한 코드는 아래와 같다.

코드

```
for k, v in name_age.items():
    if v >= 20:
        adult_names["adult"].append(k)
    else:
        adult_names["non-adult"].append(k)
```

```
for k, v in name_age.items():
    if v >= 20:
        key = "adult"
    else:
        key = "non-adult"
    adult_names[key].append(k)
```

```
for k, v in name_age.items():
    key = "adult" if v >= 20 else "non-adult"
    adult_names[key].append(k)
```

```
for k, v in name_age.items():
    adult_names["adult" if v>=20 else "non-adult"].append(k)
```

문제 7) 문자열로 구성된 리스트 strings와, 문자열-문자열 키-밸류로 이루어진 딕셔너리 options가 있다. 이 변수들에 대하여, 아래의 문제들을 해결해 보자.
④ 문제 ③에서 의도한 대로, 아래 코드의 출력 결과와 같이 strings의 각 문자열 사이에 options의 "sep"키에 대응하는 밸류 문자열을 끼워넣기 위해 빈칸에 올 수 있는 코드로 적절한 것은?

코드

```
strings = ["python", "programming"]
options = {"sep": "/"}
join_strings = ""
for i in range(len(strings)):
    (          )
    join_strings += s + strings[i]
print(join_strings)
```

실행 결과

```
python/programming
```

보기

ⓐ
```
if i == 0:
    s = options.get("sep", "")
    join_strings += strings[i]
    continue
```

ⓑ
```
if i == 0:
    s = options.get("sep", "")
    join_strings += s
    continue
```

ⓒ
```
if i == 0:
    s = options.get("sep", "")
    join_strings += strings[i]
    break
```

ⓓ
```
if i == 0:
    s = options.get("sep", "")
    join_strings += s
    break
```

해설

위 문제의 코드는 options["sep"]을 strings의 각 문자열 사이에 끼워넣은 문자열을 구하는 코드이다. 이 기능을 다른 방식으로 구현한 코드는 아래와 같다.

코드

```
strings = ["python", "programming"]
options = {"sep": "/"}
join_strings = ""
s = options.get("sep", "")
for i in range(len(strings)):
    if i == 0:
        join_strings += strings[i]
    else:
        join_strings += s + strings[i]
print(join_strings)
```

```
strings = ["python", "programming"]
options = {"sep": "/"}
join_strings = ""
s = options.get("sep", "")
for i in range(len(strings)):
    join_strings = join_strings + (s if i else "") + strings[i]
print(join_strings)
```

```
s = options.get("sep", "")
join_strings = s.join(strings)
```

```
join_strings = options.get("sep", "").join(strings)
```

챕터 10

문제 4) 200개의 jpg 사진 파일들의 이름을 문자열 요소로 하는 리스트 data가 있다. 사진 파일들의 특징이 아래와 같을 때, 각 문제를 해결해 보자.

· 짝수 번호의 파일은 개 사진, 홀수 번호의 파일은 고양이 사진 파일이다.

· 각 파일의 번호는 1에서 200까지 data에 숫자 크기순으로 정렬되어 있다. 한 자리, 두 자리 수에는 각각 00, 0이 채워져 있다. 예를 들어, 2번 파일의 이름은 002.jpg, 45번 파일의 이름은 045.jpg, 150번 파일의 이름은 150.jpg이다.

① [객관식] 아래의 보기 중, 리스트 data를 구현한 코드로 적절한 것은?

보기	
ⓐ	data = [] for i in range(1, 201): data.append("{}.jpg".format(i))
ⓑ	data = [] for i in range(1, 201): data.append("{:3}.jpg".format(i))
ⓒ	data = [] for i in range(1, 201): data.append("{0:3}.jpg".format(i))
ⓓ	data = [] for i in range(1, 201): data.append("{:03}.jpg".format(i))

ⓓ data 리스트에 1에서 200까지 반복하여 각 숫자를 세 자리로 표현한 파일 이름을 담는다. 위 코드를 다른 방식으로 구현한 코드는 아래와 같다.

코드
data = ["{:03}.jpg".format(i) for i in range(1, 201)]

② [객관식] 딕셔너리 변수 data_dict에 파일 이름을 개/고양이를 기준으로 분리하여 저장하고 싶다. "dog" 키에는 ["002.jpg", "004.jpg", ..., "198.jpg", "200.jpg"]을, "cat" 키의 밸류 리스트에는 ["001.jpg", "003.jpg", ..., "197.jpg", "199.jpg"]를 담으려 한다. 문제의 보기 중, 아래 코드의 빈칸에 들어갔을 때 data_dict를 올바르게 구현한 것은?

코드
( )
data_dict = {"dog": dog_list, "cat": cat_list}

보기	
ⓐ	dog_list = ["{:03}.jpg".format(i) for i in range(1, 201) if i%2==0] cat_list = ["{:03}.jpg".format(i) for i in range(1, 201) if i%2==1]
ⓑ	dog_list = ["{:03}.jpg".format(i) for i in range(1, 201) if i%2==1] cat_list = ["{:03}.jpg".format(i) for i in range(1, 201) if i%2==0]
ⓒ	dog_list = ["{:03}.jpg".format(i) if i%2==0 for i in range(1, 201)] cat_list = ["{:03}.jpg".format(i) if i%2==1 for i in range(1, 201)]
ⓓ	dog_list = ["{:03}.jpg".format(i) if i%2==1 for i in range(1, 201)] cat_list = ["{:03}.jpg".format(i) if i%2==0 for i in range(1, 201)]

해설

ⓐ dog_list에는 i가 짝수인 경우, cat_list에는 i가 홀수인 경우에만 i 값에 대응되는 파일이름을 저장한다. 위 코드를 다른 방식으로 구현한 코드는 아래와 같다.

코드
dog_list = ["{:03}.jpg".format(i) for i in range(1, 201) if not i%2] cat_list = ["{:03}.jpg".format(i) for i in range(1, 201) if i%2]

dog_list, cat_list = [], []
for i in range(1, 201):
  if i%2==0: dog_list.append("{:03}.jpg".format(i))
  if i%2==1: cat_list.append("{:03}.jpg".format(i))

dog_list = ["{:03}.jpg".format(i) for i in range(2, 201, 2)] cat_list = ["{:03}.jpg".format(i) for i in range(1, 201, 2)]

문제 5) 숫자로 이루어진 리스트 nums가 있다. 반복문을 통해 nums의 요소 개수만큼 반복하여 nums의 첫 번째 요소부터 i번째 요소까지의 누적 제곱 평균을 출력하는 알고리즘을 구현하려 한다. 예를 들어, nums = [3, 6, 3, 2]의 누적 제곱 평균을 출력한 결과는 아래와 같다.

· 첫 번째 출력 : 9 / 1 = 9.0
· 두 번째 출력 : (9 + 36) / 2 = 22.5
· 세 번째 출력 : (9 + 36 + 9) / 3 = 18.0
· 네 번째 출력 : (9 + 36 + 9 + 4) / 4 = 14.5

④ 매 반복마다 d_sum 변수에 제곱 값을 더한 후 반복 인덱스 값으로 나누려 한다. 문제의 알고리즘을 적절하게 구현하기 위해 아래의 코드의 빈칸에 들어갈 코드로 적절한 것은?

코드

```
nums = [3, 6, 3, 2]
d_sum = 0.0
( ⓐ / ⓑ / ⓒ / ⓓ )
```

보기	

ⓐ	for i, n in enumerate(nums, 1):
	  d_sum += n*n
	  mean_d = d_sum / i
	  print(mean_d)

ⓑ	for i, n in enumerate(nums):
	  d_sum += n*n
	  mean_d = d_sum / i
	  print(mean_d)

ⓒ	for i, n in enumerate(nums):
	  d_sum += n*n
	  mean_d = d_sum / i+1
	  print(mean_d)

ⓓ	for i, n in enumerate(nums):
	  d_sum += n*n
	  mean_d = d_sum // i+1
	  print(mean_d)

해설

ⓐ 매 반복마다 n의 제곱의 누적 합 평균을 출력한다. ⓐ의 코드를 다른 방식으로 구현한 코드는 아래와 같다.

코드

for i, n in enumerate(nums, 1):
    d_sum += n**2
    mean_d = d_sum / i
    print(mean_d)

for i, n in enumerate(nums):
    d_sum += n*n
    mean_d = d_sum / (i+1)
    print(mean_d)

for i in range(len(nums)):
    d_sum += nums[i]*nums[i]
    mean_d = d_sum / (i+1)
    print(mean_d)

문제 6) 챕터 6의 예제에서 구한 소수(prime number) 판별 알고리즘을 리스트 컴프리헨션으로 간결하게 구현하려 한다.

③ [객관식] 소수는 1과 n 이외의 모든 수에 대해 나누어떨어지지 않아야 한다. 이를 이용하여 n이 소수인/소수가 아닌 경우 n_is_prime이 각각 True/False가 되도록 코드를 작성하려 한다. 아래의 빈칸에 올 수 있는 코드로 적절한 것은?

문제	보기
n = 31 n_is_prime = ( )	ⓐ all([n%i for i in range(2, n)])
	ⓑ any([n%i for i in range(2, n)])
	ⓒ all([i%n for i in range(2, n)])
	ⓓ any([i%n for i in range(2, n)])

해설

ⓐ 함수 값은 n이 소수이면 True, n이 소수가 아니면 False이다. 이를 다른 방식으로 구현한 코드는 아래와 같다.

코드
not any([n%i==0 for i in range(2, n)])

챕터 11

문제 4) 이번 챕터에서 short circuit evaluation 및 시간 복잡도 개념에 대하여 학습하였다. 이를 바탕으로 직전 챕터의 연습문제에서 구현한 소수 판별 알고리즘을 효율적으로 구현하려 한다. 아래 문제들을 해결하여 소수 판별 알고리즘을 한 줄로 구현하는 코드를 작성해 보자.

③ [객관식] 아래 보기의 코드 중, 소수 판별 알고리즘을 적절하게 구현한 것은?

보기	
ⓐ	num = 901256437 is_prime = any(num%i==0 for i in range(2, int(num**(1/2))+1)) print(is_prime)
ⓑ	num = 901256437 is_prime = all(num%i==0 for i in range(2, int(num**(1/2))+1)) print(is_prime)
ⓒ	num = 901256437 is_prime = not all(num%i==0 for i in range(2, int(num**(1/2))+1)) print(is_prime)
ⓓ	num = 901256437 is_prime = not any(num%i==0 for i in range(2, int(num**(1/2))+1)) print(is_prime)

해설

ⓓ 2부터 num의 제곱근 숫자까지 반복하여 반복 숫자로 나누어떨어지는지 검사하는 방식으로 소수를 검사한다. 챕터 10의 연습문제 4 추가 정답의 알고리즘의 방식을 사용하였다. 위 코드를 다른 방식으로 구현한 코드는 아래와 같다.

코드
not any([n%i==0 for i in range(2, n)])

챕터 12

문제 8) 주어진 입력 중에서 두 번째로 큰 값을 리턴하는 함수를 설계하려 한다. 아래의 문제들을 해결하여 의도한 함수를 설계해 보자.

② [객관식] 함수 내에서 n1, n2, 그리고 튜플의 요소로 이루어진 리스트를 total 변수로 선언 후 해당 리스트의 최댓값을 max_total 변수에 대입하려 한다. 아래 보기 중 이를 구현한 코드로 적절한 것은?

보기	
ⓐ	total = nums + n1 + n2 max_total = max(total)
ⓑ	total = list(nums) + [n1, n2] max_total = max(total)
ⓒ	total = list(nums + n1 + n2) max_total = max(total)
ⓓ	total = list(nums) + list(n1, n2) max_total = max(total)

해설

ⓑ nums의 모든 요소와 n1, n2를 담은 리스트를 total에 대입한 후, total 리스트 내 최댓값을 구한다. 이 코드를 다른 방식으로 구현한 코드는 아래와 같다.

코드
total = [*nums, n1, n2] max_total = max(total)

③ [객관식] 문제 ②에서 구한 max_total은 전체 값 중 최댓값이다. total에서 이를 제거한 후에 다시 total의 최댓값을 return하면 두 번째로 큰 값을 리턴할 수 있다. 아래 보기 중 이를 구현한 코드로 적절한 것은?

보기	
ⓐ	total.pop(max_total) return max(total)
ⓑ	total = total.pop(max_total) return max(total)
ⓒ	total = total.remove(max_total) return max(total)
ⓓ	total.remove(max_total) return max(total)

해설

위 문제의 정답은 ⓓ로, total 리스트에서 최댓값을 제거한 후에 total의 최댓값을 리턴하는 방식으로 두 번째로 큰 값을 구한다. 두 번째로 큰 값을 리턴하는 알고리즘을 다른 방식으로 구현한 코드는 아래와 같다.

코드
return sorted(total)[-2]
total.sort() return total[-2]

챕터 15

문제 5) 내가 개발한 알고리즘의 동작 시간을 측정하는 기능 time_measure를 contextmanager로 구현하려 한다. 아래의 문제를 해결하여 time_measure를 완성해 보자.

② [객관식] 아래와 같이 contextmanager를 데코레이터로 사용하여 time_measure를 설계하였다. 아래의 코드에 대한 설명으로 적절하지 않은 것은?

코드

```
from contextlib import contextmanager
from time import time
@contextmanager
def time_measure(do_print=False):
    start = time()
    yield
    end = time()
    if do_print:
        print(end-start)
```

해설

위 코드의 time_measure를 with문에 사용할 경우 with문 내부가 실행되는 동안의 시간을 측정할 수 있다. 위 코드를 다른 방식으로 구현한 코드는 아래와 같다.

코드

```
from contextlib import contextmanager
from time import time
@contextmanager
def time_measure(do_print=False):
    start = time()
    yield
    if do_print: print(time() - start)
```