BAF515 금공프3 Quiz 1

20249433 MFE 최재필

수업 중에 배운 내용만을 이용해서 코드를 작성해주시면 됩니다.

사용자 정의함수, 조건문, 반복문 등 아직 배우지 않은 내용을 활용하면 안됩니다.

In [ ]:
```
import math
```

1.

Calculate the following

(1)

(1) The roots of the quadratic equation $ax^2 + bx + c = 0$ is $\frac{-b \pm \sqrt{b^2-4ac}}{2a}$. Find the roots when a = $2, b = -1, c = -15$.

In [ ]:
```
a = 2
b = -1
c = -15

roots = [
    (-b + math.sqrt(b**2 - 4*a*c)) / (2*a),
    (-b - math.sqrt(b**2 - 4*a*c)) / (2*a),
    ]

roots
```

Out[ ]:
```
[3.0, -2.5]
```

(2)

(2) Consider the Normal pdf function $f(x;\mu,\sigma^2)=\frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$ with the parameters $\mu=2,\sigma^2=3$. What is the value of $f(x=1)$.

In [ ]:
```
mu = 2
variance = 3

x = 1

f = (1 / (math.sqrt(2 * math.pi * variance))) * math.exp(-((x - mu)**2) / (2 * variance))
f
```

Out[ ]:
```
0.19496965572274116
```

2.

Briefly explain why the error occur in the following expression

(1)

(1)

```
>>> a = input("enter a number:")

enter a number:5

>>> a+3
```

In [ ]:
```
a = input('enter a number')
```

In [ ]:
```
a
```

Out[ ]:
```
'5'
```

In [ ]:
```
a + 3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[123], line 1
----> 1 a + 3

TypeError: can only concatenate str (not "int") to str
```

As shown above, it's because `input()` function takes the input as `str`, and `str` cannot add with 3, which is an `int`.

To correct this:

In [ ]:
```
int(a) + 3
```

Out[ ]:
```
8
```

(2)

(2)

>>> tmp = 'My String'

>>> tmp[10]

In [ ]:
```
tmp = 'My String'
```

In [ ]:
```
tmp[10]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[126], line 1
----> 1 tmp[10]

IndexError: string index out of range
```

As shown above, it's because string index is out of range. The given string, `tmp` only has 0~8 indices (length: 9)

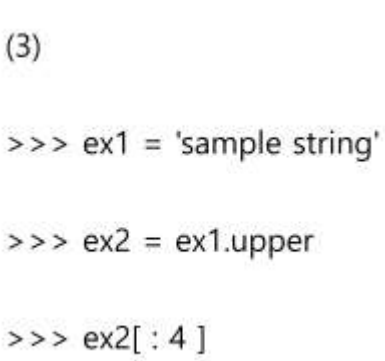
In [ ]:
```
len(tmp)
```

Out[ ]:
```
9
```

In [ ]:
```
tmp[8]
```

Out[ ]:
```
'g'
```

(3)



In [ ]:
```
ex1 = 'sample string'
ex2 = ex1.upper
```

In [ ]:
```
ex2[:4]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[130], line 1
----> 1 ex2[:4]

TypeError: 'builtin_function_or_method' object is not subscriptable
```

This error occurs because `.upper()` method was not called properly. It should have `()` in the end. If `()` is missing, `.upper` is the method function itself.

In [ ]:
```
type(ex2)
```

Out[ ]:
```
builtin_function_or_method
```

To correct this:

In [ ]:
```
ex2 = ex1.upper()

ex2[:4]
```

Out[ ]:
```
'SAMP'
```

3.

Create the following string object 'grade'

3. Create the following string object 'grade'.

```
grade='ABCDF'
```

In [ ]:
```
grade = 'ABCDF'
```

(1)

Using the `+` operator on `grade`, create `grade_str` as follows.

```
>> grade_str

'ABCDFFFDCBA'
```

In [ ]:
```
grade_str = grade + 'F' + grade[::-1]

grade_str
```

Out[ ]:
```
'ABCDFFFDCBA'
```

(2)

Count the number of 'A' in `grade_str`

In [ ]:
```
list(grade_str).count('A')
```

Out[ ]:
```
2
```

(3)

Present 4 different slicing expressions to extract 'FFF' in `grade_str`

In [ ]:
```
# 1 Normal Slicing
grade_str[4:7]
```

Out[ ]:
```
'FFF'
```

In [ ]:
```
# 2 Using negative step
grade_str[6:3:-1]
```

Out[ ]:
```
'FFF'
```

In [ ]:
```
# 3 Using negative indices
grade_str[-7:-4]
```

Out[ ]:
```
'FFF'
```

In [ ]:
```
# 4 Using negative indices and negative step
grade_str[-5:-8:-1]
```

Out[ ]:
```
'FFF'
```

(4)

Modify `grade_str` in (3) as the following

```
>> grade_str

ABCDAAADCBA
```

In [ ]:
```
grade_str = grade_str.replace('FFF', 'AAA')
grade_str
```

Out[ ]:
```
'ABCDAAADCBA'
```

(5)

Change all letters of `grade_str` to lower case

In [ ]:
```
grade_str.lower()
```

Out[ ]:
```
'abcdaaadcba'
```

4.

Briefly explain why the error occurs in the following expression.

(1)

(1) >>> L = [[1,3,5,7,9], [2,4,6,8,10]]

>>> L[0][1:2]=30

In [ ]:
```
L = [
    [1, 3, 5, 7, 9],
    [2, 4, 6, 8, 10],
    ]
```

In [ ]:
```
L[0][1:2] = 30
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[143], line 1
----> 1 L[0][1:2] = 30

TypeError: can only assign an iterable
```

This error occurs because an `int` value instead of iterable(`list`) was assigned.

In [ ]:
```
L[0][1:2]
```

Out[ ]:
```
[3]
```

To correct this:

In [ ]:
```
L[0][1:2] = [30]
```

In [ ]:
```
L
```

Out[ ]:
```
[[1, 30, 5, 7, 9], [2, 4, 6, 8, 10]]
```

(2)

(2) >>> T=(10, 20, 30)

>>> T[:2]+(40)

In [ ]:
```
T = (10, 20, 30)
```

In [ ]:
```
T[:2] + (40)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[148], line 1
----> 1 T[:2] + (40)

TypeError: can only concatenate tuple (not "int") to tuple
```

This error occurs because `(40)` was interpretted as `int` instead of `tuple`

To correct this:

In [ ]:
```
T[:2] + (40,)
```

Out[ ]:
```
(10, 20, 40)
```

(3)

(3) >>> D = {'A':10, 'B':20, 'C':30}

>>> D2=D

>>> del D2['A']

>>> D['A']

In [ ]:
```
D = {
    'A': 10,
    'B': 20,
    'C': 30,
    }
```

In [ ]:
```
D2 = D
```

In [ ]:
```
del D2['A']
```

In [ ]:
```
D['A']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[153], line 1
----> 1 D['A']

KeyError: 'A'
```

This error occurs because `D`'s key `'A'` has already been deleted.

Although variable `D2 = D` was declared, it is merely referencing D object in the memory.

Thus, the change in the `D2` object itself is shown in `D`.

To correct this:

In [ ]:
```
D = {
    'A': 10,
    'B': 20,
    'C': 30,
    }
```

In [ ]:
```
D2 = D.copy()
```

In [ ]:
```
del D2['A']
```

In [ ]:
```
D['A']
```

Out[ ]:
```
10
```

In [ ]:
```
D
```

Out[ ]:
```
{'A': 10, 'B': 20, 'C': 30}
```

In [ ]:
```
D2
```

Out[ ]:
```
{'B': 20, 'C': 30}
```

(4)

(4) >>> D3={['Park','male']:30, ['Lee','female']:28, ['Kim','male']:34 }

In [ ]:
```
D3 = {
    ['Park', 'male']: 30,
    ['Lee', 'female']: 28,
    ['Kim', 'male']: 34,
}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[160], line 1
----> 1 D3 = {
      2     ['Park', 'male']: 30,
      3     ['Lee', 'female']: 28,
      4     ['Kim', 'male']: 34,
      5 }

TypeError: unhashable type: 'list'
```

This error occurs because the dictionary's key is unhashable type, `list`.

To correct this:

In [ ]:
```
D3 = {
    ('Park', 'male'): 30,
    ('Lee', 'female'): 28,
```

```
    ('Kim', 'male'): 34,
}
```

(5)

(5) >>> dict_y = { (1,) : 10, (2,): 20, (3,): 30, (4,): 40 }

>>> dict_y[-2 :]

In [ ]:
```
dict_y = {
    (1,): 10,
    (2,): 20,
    (3,): 30,
    (4,): 40,
}
```

In [ ]:
```
dict_y[-2:]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[163], line 1
----> 1 dict_y[-2:]

TypeError: unhashable type: 'slice'
```

This error occurs because dictionary, although it maintains its order from Python 3.7+, is not indexable nor subscriptible.

To correct this: (Although it is not common/recommended to slice dictionary key-value pairs,)

In [ ]:
```
list(dict_y)[-2:] # Only maintains the keys
```

Out[ ]:
```
[(3,), (4,)]
```

In [ ]:
```
list(dict_y.items())[-2:] # Maintains the key-value pairs
```

Out[ ]:
```
[((3,), 30), ((4,), 40)]
```

5.

Create the following list object 'days'.

days = ['Mon', 'Tues', 'Wed', 'Thur', 'Fri', ['Sat', 'Sun']]

In [ ]:
```
days = ['Mon', 'Tues', 'Wed', 'Thur', 'Fri', ['Sat', 'Sun']]
```

(1)

Extract elements from `days` as shown below

① [['Sat', 'Sun']]

② [['Sat', 'Sun'], 'Thur', 'Tues']

③ 'Sat'

In [ ]:
```
# 1

days[-1:]
```

Out[ ]:
```
[['Sat', 'Sun']]
```

In [ ]:
```
# 2

days[-1:-6:-2]
```

Out[ ]:
```
[['Sat', 'Sun'], 'Thur', 'Tues']
```

In [ ]:
```
# 3

days[-1][0]
```

Out[ ]:
```
'Sat'
```

(2)

Modify `days` as shown below by applying the slicing(`:`) and concatenation operator(`+`) and name it `days2`

```
>>> days2

[['Mon', 'Tues', 'Wed', 'Thur', 'Fri'], 'Sat', 'Sun']
```

In [ ]:
```
days2 = [days[:5]] + days[-1]
days2
```

Out[ ]:
```
[['Mon', 'Tues', 'Wed', 'Thur', 'Fri'], 'Sat', 'Sun']
```

(3)

Modify `days2` in (2) as follows, by removing the 2 items 'Wed' and 'Fri'.

```
>> days2

[['Mon', 'Tues', 'Thur'], 'Sat', 'Sun']
```

In [ ]:
```
days2[0].remove('Wed')
days2[0].remove('Fri')
```

In [ ]:
```
days2
```

Out[ ]:
```
[['Mon', 'Tues', 'Thur'], 'Sat', 'Sun']
```

(4)

Modify `days2` in (3) as following, by inserting 'W' at the given position.

```
>> days2

[['Mon', 'Tues', 'W', 'Thur'], 'Sat', 'Sun']
```

In [ ]:
```
days2[0].insert(2, 'W')
```

In [ ]:
```
days2
```

Out[ ]:
```
[['Mon', 'Tues', 'W', 'Thur'], 'Sat', 'Sun']
```

6.

Create the following list object `Nums`.

```
Nums=[1, 5, 2, 7, 3, 6, 4]
```

In [ ]:
```
Nums = [1, 5, 2, 7, 3, 6, 4]
```

(1)

Append the largest element of `Nums` to the end of `Nums`.

```
>> Nums

[1, 5, 2, 7, 3, 6, 4, 7]
```

In [ ]:
```
Nums = Nums + [max(Nums)]
Nums
```

Out[ ]:
```
[1, 5, 2, 7, 3, 6, 4, 7]
```

(2)

Sort the elements in `Nums` in decreasing order.

```
>> Nums

[7, 7, 6, 5, 4, 3, 2, 1]
```

In [ ]:
```
Nums.sort(reverse=True)
```

In [ ]:
```
Nums
```

Out[ ]:
```
[7, 7, 6, 5, 4, 3, 2, 1]
```

(3)

Modify the `Nums` in (2) as the following. (Replace the 1st, 3rd, 5th and 7th elements in `Nums` with 'a'.)

```
>> Nums

['a', 7, 'a', 5, 'a', 3, 'a', 1]
```

In [ ]:
```
Nums[0::2] = ['a']*4
```

In [ ]:
```
Nums
```

Out[ ]:
```
['a', 7, 'a', 5, 'a', 3, 'a', 1]
```

7.

Create the following tuple object 'price'.

price = (180, 130, 110, 160, 140, 170)

In [ ]:
```
price = (180, 130, 110, 160, 140, 170)
```

(1)

Sort the items of `price` in ascending order so that `price` is displayed as below.

```
>> price
```

(110, 130, 140, 160, 170, 180)

In [ ]:
```
price = tuple(sorted(price))
price
```

Out[ ]:
```
(110, 130, 140, 160, 170, 180)
```

(2)

Write a code that returns `True` if `price` has the value 170 and `False` otherwise.

In [ ]:
```
170 in price
```

Out[ ]:
```
True
```

(3)

Insert 3 zeros instead of 5th value 160 in `price`, so that `price` is displayed as below.

>> price

(110, 130, 140, 160, 0, 0, 0, 180)

In [ ]:
```
price[:4] + (0, 0, 0) + price[-1:]
```

Out[ ]:
```
(110, 130, 140, 160, 0, 0, 0, 180)
```