【模擬】 Pythonカテスト

合計点 70/70 点

このテストは，Python3.9.13，BeautifulSoup4.11.1，Selenium4.6.0を想定している.

このフォームの送信時に回答者のメールアドレス（**c0a2004727@edu.teu.ac.jp**）が記録されました。

✓ 次のコードを実行した際の出力またはエラーとして適切なものを選びなさい. 1/1

```
x = tuple(range(10))
n = None
if n:
    print(len(x))
else:
    x[0] = 1
    print(x[0])
```

- ○ 10
- ○ 1
- ◉ TypeError ✓
- ○ AttributeError

✓ 次のコードを実行した際，7行目と8行目の出力として適切なものを選びなさい. 1/1

```
1 x = list(range(1, 11))
2 y = [str(i) for i in range(1, 11)]
3 x.sort()
4 y.sort()
5 z = [int(i) for i in y]
6 print(x, y)
7 print(x == z) #1
8 print(y == z) #2
```

- ○ どちらもTrue
- ◉ どちらもFalse ✓
- ○ 7行目だけTrue
- ○ 8行目だけTrue

✓ 次のコードを実行した際の出力として適切なものを選びなさい. 1/1

```
1 x = list(range(10))
2 y = x.sort()
3 print(type(y))
```

- ○ list
- ○ type
- ○ sort
- ◉ NoneType ✓

✓ 次のコードを実行した際に発生するエラーの原因として適切なものを選びなさい. 1/1

```
1  class A:
2      def __init__(self, a, b, c=3):
3          self.a = a
4          self.b = b
5          self.c = c
6
7  class A_d(A):
8      def __init__(self, a, b=2):
9          super.__init__(a, b)
10
11
12 q = A_d(1, 5)
13 print(q.b)
```

- ○ クラスA_dのインスタンスを生成する際，2つの引数を指定しているから
- ○ クラスA_dは抽象クラスであるクラスAを継承しているから
- ○ クラスAのイニシャライザのパラメータ数とクラスA_dのイニシャライザのパラメータ数が異なるから
- ◉ 9行目で親クラスのイニシャライザを呼び出す方法が間違っているから ✓

✓ 次のコードを実行した際の出力として適切なものを選びなさい. 1/1

```python
class A:
    def __init__(self):
        self.a = 0
    def __str__(self):
        return f"{self.a+1}"
    def __repr__(self):
        return f"{self.a+2}"


a=A()
print(a)
```

- a
- 0
- 1 ✓
- 2

✓ 次のコードを実行した後に，Falseとなるものを**すべて**選びなさい． 1/1

```
a = [i%2 for i in range(10)]
b = [0 if i%2==0 else 1 for i in range(10)]
c = [i for i in range(2) for _ in range(5)]
d = list(range(2))*5
```

- [ ] a == b
- [x] a == c ✓
- [ ] a == d
- [x] b == c ✓
- [ ] b == d
- [x] c == d ✓

✓ 次のコードを実行した際に発生するエラーを選びなさい．エラーが発生しない場合は「発生しない」を選びなさい． 1/1

```
x_s = [tuple, set, str]
for x in x_s:
    a = [n for n in x(range(10))]
```

- (●) 発生しない ✓
- ( ) ValueError
- ( ) TypeError
- ( ) IndexError

✓ 内包表記のみを用いた定義ができないものを選びなさい. （内包表記のあとに別の型に変換することを除く） 1/1

- list
- tuple
- dict
- set

✓ itertoolsモジュールに含まれない関数を選びなさい. 1/1

- accumulate
- groupby
- slice
- islice

✓ 次のコードを実行した際の出力として適切なものを選びなさい. 1/1

```
1 obj_list = [1.0, 1, 1/5, [], (), {}, int, float]
2 obj_set = set([type(o)for o in obj_list])
3 print(len(obj_set))
```

- 5
- 6
- 7
- 8

✓ 次のコードを実行した際，3行目と4行目の出力として適切なものを選びなさい. 1/1

```
1 a = [i for i in range(0, 100, 5)]
2 b = [i for i in range(100) if i%5==0]
3 print(a == b) #1
4 print(a is b) #2
```

- ○ どちらもTrue
- ○ どちらもFalse
- ◉ 3行目だけTrue ✓
- ○ 4行目だけTrue

✓ 次のコードを実行した際の出力として適切なものを選びなさい. 1/1

```python
from dataclasses import dataclass

@dataclass
class D:
    a :int
    b :float
    c :str


data = D(1, 2, 3)
print(type(data.c))
```

- (●) int ✓
- ( ) float
- ( ) str
- ( ) D

✓ seleniumのwebdriverを用いて，次のQ.htmlから2つ目のハイパーリンクのみを取得する際の適切なコードを選びなさい. 1/1

Q.html

```html
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <title>HTML Sample</title>
</head>
<body>
    <h1>見出し</h1>
    <p>本文</p>
    <p name="link1">1</p>
    <a id="12hk21" href="http://sample1.com">link1</a>
    <p name="link2">2</p>
    <a id="jkoe09" href="http://sample2.com">link2</a>

</body>
</html>
```

- ○ driver.find_element(By.TAG_NAME, "a")
- ○ driver.find_element(By.NAME, "link2")
- ◉ driver.find_element(By.LINK_TEXT, "link2") ✓
- ○ driver.find_element(By.LINK_TEXT, "link1").next()

✓ 辞書から値だけを抽出したビューを返すメソッドを選びなさい. 1/1

- ○ value()
- ○ item()
- ○ key()
- ◉ values() ✓
- ○ items()
- ○ keys()

✓ 次のコードを実行した際，出力が「**234**」となった．空欄に当てはまるコードとして適切なものを選びなさい． 1/1

```
lst = [2, 4, 3]

[空欄]

for num in lst:
    print(num, end="")
print()
```

- (●) lst.sort() ✓
- ( ) sort(lst)
- ( ) lst.sorted()
- ( ) sorted(lst)

✓ 次のコードを実行した際に発生するエラーを選びなさい． 1/1

```
def change_num(nums):
    nums[0] = nums[0] // nums[1] - nums[1]


numbers = (0, 1)
change_num(numbers)
print(numbers[0]+numbers[1]+numbers[2])
```

- ( ) IndexError
- (●) TypeError ✓
- ( ) ZeroDivisionError
- ( ) AttributeError

✓ クラスHumanに，人間の名前を表す属性nameを作成し，これを取得・設定するプロパティを作成した．ゲッタとセッタのデコレータの組み合わせとして正しいものを選びなさい． 1/1

- ○ ゲッタ：@name.getter／セッタ：@name.setter
- ◉ ゲッタ：@property／セッタ：@name.setter
- ○ ゲッタ：@name.getter／セッタ：@property
- ○ ゲッタ：@property.name／セッタ：@property.setter
- ○ ゲッタ：@property.getter／セッタ：@property.name

✓ 「__name__」の説明として正しいものを選びなさい． 1/1

- ○ インスタンスの元となるクラスを表す
- ◉ モジュールの完全修飾名や実行プログラムのエントリーポイントか否か，クラス名などを表す
- ○ インタプリタセッションでオブジェクトを調べたり，オブジェクトを含むコンテナをprintするときに動作する
- ○ インスタンス作成時に必ず動作する

✓ 次のコードを実行した際，7行出力された．6行目に出力されるものを選びなさい． 1/1

```python
import copy

class Robot:
    latest_version = 0

    def __init__(self, type_):
        self._type = type_
        Robot.latest_version += 1
        self.version = Robot.latest_version

    def version_up(self):
        Robot.latest_version += 1
        self.version = Robot.latest_version

robots = [Robot(["家事用"])]
robots.append(robots[0])
robots.append(copy.copy(robots[0]))
robots.append(copy.deepcopy(robots[0]))

robots[0].version_up()
robots[0]._type.append("娯楽用")

robots.append(robots[2])
robots.append(copy.copy(robots[2]))
robots.append(copy.deepcopy(robots[2]))

robots[2].version_up()
robots[2]._type.append("工場用")

for rbt in robots: print(rbt.version, rbt._type)
```

- 1 ['家事用']
- 2 ['家事用']
- 3 ['家事用']
- 1 ['家事用', '娯楽用']
- 2 ['家事用', '娯楽用']
- 3 ['家事用', '娯楽用']
- ◉ 1 ['家事用', '娯楽用', '工場用'] ✓
- 2 ['家事用', '娯楽用', '工場用']

3 ['家事用', '娯楽用', '工場用']

✓ with構文を使わないでファイルを読み込む際の注意点を選びなさい. 1/1

- ファイルのディスクリプタを直接扱うことになるので，環境依存のコードになる.
- ✓ ファイルが自動的に閉じられないため，忘れると他のプログラムで不具合を引き起こす危険性がある.
- ファイルの文字コードを自動的に判別してくれないため，手動で設定しなければならない.
- ファイルの読み込み・書き込みのようなモードを自動的に判別しないので，手動で設定しなければならない.

✓ 次のような内容のタブ区切りのテキストファイルを開き， ファイル全体を変数senに読み込む. このとき，2行目の文を抽出するコードとして適切なものを選びなさい. なお，改行文字は「\n」，タブ文字は「\t」である. 1/1

```
ID	投稿時刻	共有数	内容
0	41	50	C++23には添え字演算子の提案を採用してほしいな。
1	42	1000	Unicodeの対応をもっと改善しておくれ・・・
2	93	150	それよりも、コルーチンや契約の機能の方が先だ！！
```

- sen.split('\n')[-1]
- ✓ sen.split('\n')[-3]
- sen.split('\n')[0]
- sen.split('\t')[-3]

✓ 「__init__」の役割として適切なものを選びなさい. 1/1

- ○ クラスを関数のように振る舞わせるためのメソッドである.
- ◉ オブジェクトの初期化時に実行されるメソッドであり，主にメンバ変数の作成などを記述する際に使用する.
- ○ オブジェクトの削除時に実行されるメソッドであり，例えば手動でメモリ解放など特殊な処理を記述する際に使用する.
- ○ コードが読み込まれた時に必ず実行される，クラス全体を初期化するためのメソッドである.

✓ 一行で記述できるような簡単な関数や一時的な関数を作る機能として適切なものを選びなさい. 1/1

- ◉ ラムダ式
- ○ クロージャ
- ○ 関数オブジェクト
- ○ ローカル関数

✓ 次のコードの関数に対して，「**var = func**」と「**var = func(0)**」の違いを正しく述べたものを選びなさい. 1/1

```
1 def func(a):
2     return a**2
```

- funcに括弧をつけない場合，引数に乱数が暗黙的に代入されるので，括弧が無い方のvarには何かの値を2乗したものが代入される.
- funcに括弧をつけない場合，文法的に誤っているので，括弧が無い方の記述はエラーとなり実行できない.
- ◉ funcに括弧をつけない場合，varにはfunc本体が代入されるので，「var(0)」のような記述ができる. ✓
- funcに括弧をつけない場合，未定義の動作が発生するので何が起きるか分からない.

✓ 型アノテーションについて正しく述べたものを選びなさい. 1/1

- 変数の初期化をその型の値で行う.
- ◉ 変数に入る値の種類や関数が返す値の種類を示すヒントである. ✓
- 変数の宣言を静的に行うための機能であり，その型以外の値を代入できないようにする.
- 型アノテーションを使う場合は，必ず代入を行わなければならない.

✓ 次のコードを実行した際の出力として適切なものを選びなさい. 1/1

```python
def divide(func):
    def _func(a):
        return func(a) // 3
    return _func

def multiply(func):
    def _func(a):
        @divide
        def _calculate(a):
            return a * 10
        return _calculate(func(a))
    return _func

@multiply
def caluclate(a):
    return a ** 2

print(caluclate(9))
```

- ○ 81
- ○ 54
- ◉ 270 ✓
- ○ 構文エラーにより，計算できない

✓ 「**print(A("Fushimi lab"))**」を実行したら，「**Welcome to Fushimi lab!!**」と出力された．
クラスAに定義されている特殊メソッドを選びなさい．

- ○ __call__()
- ◉ __repr__()
- ○ __print__()
- ○ __next__()

次のコードのクラスPersonの属性ageとnameに対する処理のうち，エラーが発生するものを選びなさい. 1/1

```
class PersonDescriptor:
    def __get__(self, obj, ojtype=None):
        return self.value
    def __set__(self, obj, value):
        if type(value) is int:
            if value < 0:
                raise ValueError('')
            self.value = value
        elif type(value) is str:
            self.value = value
        else:
            raise TypeError('')

class Person:
    age = PersonDescriptor()
    name = PersonDescriptor()

person = Person()
```

- person.age = "I don't know"
- person.name = 0
- person.age = 150000
- person.age = 5.0 ✓

✓ dataclassについて正しく述べたものを選びなさい. 1/1

- 比較演算に関する特殊メソッドが自動的に定義されないので，必要な場合は自分で定義しなければならない.
- インスタンス変数のアドレスが共有されるので，低レベルな操作に向いている.
- (●) 算術演算に関する特殊メソッドが自動的に定義されないので，必要な場合は自分で定義しなければならない.
- メソッドを定義することができない.

✓ 次のうち，正しい記述を選びなさい．ただし，リスト，集合，辞書，タプルは組み込み型を用いる. 1/1

- 空のリスト（list）10個からなる集合（set）を作ることができる.
- (●) タプル（tuple）を辞書（dict）のキーとして使うことができる.
- intやfloatなどの数値をfor-in文のinの後ろに置くことができる.
- 集合（set）を別の集合（set）の要素とすることができる.

✓ 伏見が嫌いな（＝パイソニック**でない**）書き方を**すべて**選びなさい. 1/1

- [x] for i in range(len(monsters)): print(monsters[i])
- [x] for i in monsters: print(i)
- [ ] for mon in monsters: print(mon)
- [x] print(monsters[0], monsters[1], monsters[2], monsters[3], monsters[4], monsters[5])

✓ zukanオブジェクトはiterableであるが，「**print(zukan[10])**」とすると「**TypeError: 'Zukan' object is not subscriptable**`」というエラーが出る．理由として適切なものを選びなさい．

- ○ __get__が定義されていないから
- ◉ __getitem__が定義されていないから
- ○ __repr__が定義されていないから
- ○ __iter__が定義されていないから

✓ 次のコードとほぼ等価なコードを選びなさい. 1/1

```
@f1(arg)
@f2
def func():
    pass
```

```
def func(arg):
    pass

func = f1(f2(func))
```

選択肢 2

```
def func():
    pass

func = f1(arg)(f2(func))
```

選択肢 1 ✓

```
def func():
    pass

func = f2()(f1(func))
```

選択肢 3

```
def func():
    pass

func = f1(arg)(f2(func()))
```

選択肢 4

✓ 次のコード（poke_name.py）をエラーなく実行するためのコマンドを選びなさい. 1/1

```python
import sys

def read_names(fp):
    省略
    return names # リスト

file_path = sys.argv[1]
row_num = int(sys.argv[2])-1
names = read_names(file_path)
print(names[row_num])
```

- ○ python poke_name.py 10 data/poke_names.txt
- ◉ python poke_name.py data/poke_names.txt 10 ✓
- ○ python poke_name.py
- ○ python poke_name.py sys.argv[1] sys.argv[2]

✓ 次の説明のうち**適切でない**ものを選びなさい. 1/1

- ○ iterableは，for-in文で繰り返し可能なオブジェクトである.
- ○ sizedは，len()関数で要素数を返すオブジェクトである.
- ◉ immutableは，作成後に値を変更できるオブジェクトである. ✓
- ○ callableは，() を付して呼び出せるオブジェクトである.

✓ 次の説明のうち**適切でない**ものを選びなさい.

1/1

- ○ strはiterableである.
- ◉ intはmutableである.
- ○ listはmutableである.
- ○ setはiterableである.

✓ 次のコードを実行すると，図のようなエラーが発生する．理由として適切なものを選びなさい.

1/1

```
1 class Monster:
2     def __init__(self, name, level):
3         self.__name = name
4         self.__level = level
5
6 m = Monster("ピカチュウ", 25)
7 print(m.__level)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [2], in <cell line: 7>()
      4         self.__level = level
      6 m = Monster("ピカチュウ", 25)
----> 7 print(m.__level)

AttributeError: 'Monster' object has no attribute '__level'
```

- ○ インスタンス変数__levelのアンダースコアが多いため
- ○ 名前マングリングにより「__level」は「_level」という名前に変わっているから
- ○ 名前マングリングにより「__level」はプライベート変数になりクラス外部からアクセスできなくなっているから
- ◉ 名前マングリングにより「__level」は「_Monster__level」という名前に変わっているから

✓ 次のコードの関数の呼び出し方として**適切でないもの**を選びなさい。 1/1

```
def func(param1, param2=None, *param3):
    pass
```

- func("hoge", "fuga", "piyo")
- func(param2="fuga") ✓
- func(param2="fuga", param1="hoge")
- func("hoge", "fuga", "piyo", "boke")

✓ 次のコードを実行した際の出力として適切なものを選びなさい。 1/1

```
def greeting(greet):
    return f"{greet} world!"

def hoge(func, *args):
    s = []
    for arg in args:
        s.append(func(arg.upper()))
    return " and ".join(s)

print(hoge(greeting, "good morning", "hello", "goodbye"))
```

- Greeting world! and Good morning world! and Hello world! and Goodbye world!
- GREETING WORLD! and GOOD MORNING WORLD! and HELLO WORLD! and GOODBYE WORLD!
- Good morning world! and Hello world! and Goodbye world!
- Good morning world! and hello world! and goodbye world!
- GOOD MORNING world! and HELLO world! and GOODBYE world! ✓
- GOOD MORNING WORLD! and HELLO WORLD! and GOODBYE WORLD!

✓ 次のコードを実行した際の出力として適切なものを選びなさい. 1/1

```python
def make_greeting(time):
    def inner1(text):
        return f"Good morning {text}!"

    def inner2(text):
        return f"Hello {text}!"

    def inner3(text):
        return f"Good evening {text}!"

    if  5 < time < 10:
        return inner1
    if 10 < time < 17:
        return inner2
    else:
        return inner3

print( make_greeting(6) )
```

- ○ <function make_greeting at 0x7fdfd035b9d0>
- ◉ <function make_greeting.<locals>.inner1 at 0x7fdfd0310c10> ✓
- ○ Good morning world!
- ○ Good morning world! Hello world! Good evening world!

✓ 組み込みデコレータ**でない**ものを選びなさい. 1/1

- ○ @property
- ○ @classmethod
- ◉ @instancemethod
- ○ @staticmethod

✓ Pythonにおいて，アンダースコアの慣習的な使い方として，適切なものを選びなさい. 1/1

- ○ 両端のダブルアンダースコア「__var__」：クラスや関数内部で使用することを明示
- ○ 先頭のシングルアンダースコア「_var」：予約語との衝突を回避
- ◉ 先頭のダブルアンダースコア「__var」：Pythonにより名前マングリングされる
- ○ シングルアンダースコアのみ「_」：Pythonにより定義された特殊なメソッド，属性
- ○ 末尾のシングルアンダースコア「var_」：必要ない値を代入するための一時的な変数

✓ ジェネレータイテレータの属性に含まれるものを**すべて**選びなさい. 1/1

- ☐ __call__()
- ☐ __getitem__()
- ☑ __iter__()
- ☐ __len__()
- ☑ __next__()

✓ シーケンスを選びなさい. 1/1

- file object
- set
- bytes
- dict

✓ 存在しない特殊メソッドを選びなさい. 1/1

- __next__
- __enter__
- __len__
- __id__

✓ 関数定義で可変長パラメータを用いた際，どの型で値を受け取るかを選びなさい. 1/1

- dict
- tuple
- list
- struct

✓ 次のコードを実行した際の出力を選びなさい．出力がない場合は「**出力なし**」を選びなさい． 1/1

```
(lambda x, y: x+y)(5, 2)
```

- (●) 7
- ( ) 5, 2
- ( ) 出力なし
- ( ) <function <lambda> at 0x00000266B504F160> ※アドレスは環境によって変わる

✓ 次のコードを実行した際の出力として，最も近いものを選びなさい． 1/1

```
dct = {"name": "fsm", "age": 45, "level": 99}
print(dct.values())
print(dct.get("test"))
print(dct.item())
```

- (●) dict_values(['fsm', 45, 99])＜改行＞None＜改行＞AttributeError
- ( ) dict_values(['fsm', 45, 99])＜改行＞KeyError＜改行＞AttributeError
- ( ) dict_values(['fsm', 45, 99])＜改行＞KeyError＜改行＞dict_items([('name', 'fsm'), ('age', 45), ('level', 99)])
- ( ) dict_keys(['name', 'age', 'level'])＜改行＞None＜改行＞dict_items([('name', 'fsm'), ('age', 45), ('level', 99)])

✓ lambda式の説明として，適切**でない**ものを選びなさい． 1/1

- (●) 複数の変数を用いることができない
- ( ) 関数オブジェクトを生成する
- ( ) 変数に代入できる
- ( ) 三項演算子が利用できる

✓ 2行目の「**eevee[-1]**」と等価なものを選びなさい． 1/1

```
1 eevee = "イーブイ"
2 eevee[-1]
```

- ( ) eevee.__get_item__(-1)
- (●) eevee.__getitem__(-1)
- ( ) eevee.__get_item__[-1]
- ( ) eevee.__getitem__[-1]

✓ 中置記法で記述された式 100 - (2 + 3) * (4 + 5) の逆ポーランド記法として，適切なものを選びなさい． 1/1

- ( ) 5 + 4 * 3 + 2 - 100
- ( ) 100 2 3 4 5 - + * +
- (●) 100 2 3 + 4 5 + * -
- ( ) - 100 * + 2 3 + 4 5

1/1

逆ポーランド記法の数式を評価する関数を実装するにあたり，利用するデータ構造として適切なものを選びなさい．

- Queue
- Stack
- Deque
- Set

✓ 次の図のように，read_main.pyでaisatsu.pyをimportしhello関数を呼び出した際，「**AttributeError: module 'aisatsu' has no attribute 'hello'**」というエラーが発生した．エラーの原因として適切なものを選びなさい． 1/1

```python
# aisatsu.py
if __name__ == "__main__":
    def hello():
        print("Hello World")
    def goodbye():
        print("GoodBye World")
```

```python
# read_main
import aisatsu

aisatsu.hello()
```

- ○ read_main.pyで，「import hello」としていないから
- ○ read_main.pyで，「import aisatsu.hello」としていないから
- ○ read_main.pyで，「hello()」としていないから
- ○ read_main.pyで，「if __name__ == "__main__"」の中で「hello()」としていないから
- ◉ aisatsu.pyで，hello関数が定義されているのが「if __name__ == "__main__"」の中だから ✓

✓ 次のコードを実行した際の出力として適切なものを選びなさい.

1/1

```
1 a, *b, c, d, e = [1, 2, 3, 4]
2 print(b)
```

- ◉ []
- ○ [2]
- ○ [1, 2]
- ○ [1, 2, 3, 4]

次のコードを実行した際の出力として，適切なものを選びなさい. 1/1

```python
def make_total():
    total = 0
    def _total():
        nonlocal total
        total += 1
        return total

    return _total

funcs = []
for i in range(5):
    funcs.append(make_total())

for fun in funcs:
    print(fun(), end=" ")
```

- 00000
- 11111 ✓
- 01234
- 12345
- 44444

✓ 次のコードを実行した際，「**インスタンス**」と出力された．空欄のコードとして適切なものを選びなさい． 1/1

```
1 class niClass:
2     __cl= "クラス"
3     def __init__(self):
4         self.__in = "インスタンス"
5 
6     def __method(self):
7         return "メソッド"
8 
9 t = niClass()
```

- ○ print(t.in)
- ○ print(t._in)
- ○ print(t.__in)
- ◉ print(t._niClass__in) ✓

✓ 次のコードを実行した際に，エラーとなるprint文を**すべて**選びなさい。 1/1

```
class Pokemon:
    def __init__(self, name, types):
        self.__name = name
        self.__types = types

    def set_name(self, name):
        self.__name = name

    def set_types(self, types):
        self.__types = types

    def get_name(self):
        if self.__name == '':
            raise ValueError
        return self.__name

    def get_types(self):
        if len(self.__types) <= 0:
            raise ValueError
        elif len(self.__types) >= 3:
            raise ValueError
        return self.__types

pokemon = Pokemon('フシギダネ', ('くさ', 'どく'))
```

- [ ] print(pokemon.get_name())
- [x] print(pokemon.name) ✓
- [ ] print(pokemon.get_types())
- [x] print(pokemon.__name) ✓
- [ ] print(pokemon._Pokemon__name)

次のコードを実行した際の出力として，適切なものを選びなさい. 1/1

```
class a:
    a = 100
    b = 200
    def __init__(self):
        self.a = a

a1 = a()
a1.a = 50
print(a.a)
```

- None
- 100 ✓
- 50
- a

✓ 3行目の空欄に入る戻り値の型アノテーションとして，適切なものを選びなさい. 1/1

```python
from typing import Iterable, Iterator

def f(_x: Iterable[int]) -> [空欄]:
    x = list(_x)
    left, right = iter(x), iter(x)
    next(right)
    return [l / r for l, r in zip(left, right)]
```

- ○ list[int]
- ◉ list[float] ✓
- ○ Iterable[int]
- ○ Iterator[float] | ZeroDivisionError

✓ 「**Hi! I am Koukaton. I am majoring in computer science.**」と1行で書かれたファイルsentence.txtを読み込み，文章から単語を抽出するプログラムを作成したい． 空欄A～Cに当てはまるものとして適切なものを選びなさい． 1/1

```
1  rfo = open('sentence.txt', 'r')
2  row = [空欄A]
3  keywords = row.rstrip().split(' ')
4  results = []
5  for keyword in keywords:
6      if keyword[-1] == '!':
7          results.append([空欄B])
8      elif keyword[-1] == '.':
9          results.append([空欄C])
10     else:
11         results.append(keyword)
12
13 print(results)
```

```
['Hi', 'I', 'am', 'Koukaton', 'I', 'am', 'majoring', 'in', 'computer', 'science']
```

- ○ 空欄A：rfo.read(3)／空欄B：keyword.rstrip('!')／空欄C：keyword.rstrip('.')
- ○ 空欄A：rfo.readlines()／空欄B：keyword.rstrip(-1)／空欄C：keyword.rstrip(-1)
- ◉ 空欄A：rfo.readline()／空欄B：keyword.rstrip('!')／空欄C：keyword.rstrip('.') ✓
- ○ 空欄A：rfo.readlines()／空欄B：keyword.split('!')[0]／空欄C：keyword.split('')[0]

✓ 2つのリストa, b を 「**is**」 で比較した際，Trueとなるa, bの特徴として，適切**でない**ものを選びなさい．

- ○ id(a)とid(b)が同じである
- ○ aの要素の値を変更するとbの要素の値も変更される
- ○ b = a によってこれらのリストは作成された
- ◉ b = copy.deepcopy(a)としてこれらのリストは作成された

✓ 次のコードを実行した際，下段の図の6行目のprint文で最初に出力されるものを選びなさい. 1/1

```python
class Zukan:
    def __init__(self):
        self.titles = ["フシギダネ", "フシギソウ", "フシギバナ",
                       "ヒトカゲ", "リザード", "リザードン",]
        self.idx = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.idx == len(self.titles):
            raise StopIteration()
        title = self.titles[self.idx]
        self.idx += 1
        return title
```

```python
zukan = Zukan()
print(next(zukan))
print(next(zukan))

for i, z in enumerate(zukan, 1):
    print(f"{i:03d}\t{z}")
```

- ○ 001 フシギダネ
- ○ 003 フシギダネ
- ◉ 001 フシギバナ ✓
- ○ 003 フシギバナ
- ○ StopIteration

✓ 「**gen = (i*i for i in range(20) if i%2 == 0)**」に対して，エラーが**出ない**print文を**すべて**選びなさい. 1/1

- [ ] print(gen[4])
- [ ] print(len(gen))
- [x] print(next(gen))
- [x] print(16 in gen)
- [ ] print(gen())

✓ 「**rng = range(20)**」に対して，エラーが**出ない**print文を**すべて**選びなさい. 1/1

- [x] print(rng[4])
- [x] print(len(rng))
- [ ] print(next(rng))
- [x] print(16 in rng)
- [ ] print(rng())

✓ 特殊メソッド「**__get__**」と「**__set__**」を持つオブジェクトをなんと呼ぶか，適切なものを選びなさい. 1/1

- ( ) イテレータ
- ( ) ジェネレータ
- (•) ディスクリプタ
- ( ) クロージャ
- ( ) コンテキストマネージャ

✓ 図の下段（special.py）を「**python special.py**」のようにコマンドラインで実行した際に，図の上段の9行目のprint文について正しい記述を選びなさい. 1/1

```python
# pokemon.py
print(f"{__file__}の{__name__}(ifの外)")

class Monster:
    def __init__(self):
        print(f"{__class__.__name__}クラスの{__class__.__init__.__name__}関数")

if __name__ == "__main__":
    print(f"{__file__}の{__name__}(ifの中)")
    m = Monster()
```

```python
# special.py
from pokemon import Monster

if __name__ == "__main__":
    print(f"{__file__}の{__name__}(ifの中)")
    m = Monster()
```

- ○ 「pokemon.pyの__main__(ifの中)」と出力される
- ○ 「special.pyの__main__(ifの中)」と出力される
- ○ 「NoneのNone(ifの中)」と出力される
- ◉ 実行されない ✓

✓ 特殊属性「**__dict__**」について正しい記述を選びなさい. 1/1

- ○ すべてのオブジェクトが持っている
- ○ インスタンスは持っているが，クラスは持っていない
- ◉ listオブジェクトは__dict__を持っていない ✓
- ○ intオブジェクトの__dict__の値は書き換え可能である

✓ 次のコードにおいて，上段3行目の「**self.attrname**」に代入される値を選びなさい. 1/1

```
class PokeDesc:
    def __set_name__(self, owner, attrname):
        self.attrname = attrname

    def __get__(self, obj, objtype=None):
        return obj.__dict__[self.attrname]

    def __set__(self, obj, value):
        obj.__dict__[self.attrname] = value
```

```
class Monster:
    title = PokeDesc()

    def __init__(self, title):
        self.title = title
```

```
m = Monster("ピカチュウ")
```

- ○ Monster
- ○ PokeDesc
- ◉ "title" ✓
- ○ "self.title"
- ○ "ピカチュウ"

✓ 次のコードを実行した後に，エラーが出るprint文を選びなさい. 1/1

```
from collections import defaultdict

class Monster:
    def __init__(self):
        pass

class Pokemon:
    def __init__(self, title):
        self.title = title

def func():
    return

gen = (i for i in range(10))

def_dct1 = defaultdict(list)
def_dct2 = defaultdict(Monster)
def_dct3 = defaultdict(Pokemon)
def_dct4 = defaultdict(func)
```

- ○ print(def_dct1[3])
- ○ print(def_dct2[3])
- ◉ print(def_dct3[3]) ✓
- ○ print(def_dct4[3])

✓ 次のコードの5行目の「**super()**」を「**self**」とした場合のエラーとして適切なものを選びなさい. 1/1

```
1  class EvenList(list):
2      def append(self, value):
3          if value%2 != 0:
4              raise ValueError(f"奇数は設定できません：{value}")
5          super().append(value)
6
7  eve_lst = EvenList()
8  for i in range(10):
9      try:
10         eve_lst.append(i)
11     except Exception as e:
12         print(e)
13 print(len(eve_lst), eve_lst)
```

```
奇数は設定できません：1
奇数は設定できません：3
奇数は設定できません：5
奇数は設定できません：7
奇数は設定できません：9
5 [0, 2, 4, 6, 8]
```

- (●) RecursionError: maximum recursion depth exceeded in comparison ✓
- ( ) AttributeError: 'EvenList' object has no attribute 'append'
- ( ) ValueError: 奇数は設定できません：1
- ( ) TypeError: not all arguments converted during string formatting