Code of Honor

I pledge to follow the Code of Honor and obey the rules for taking this test:

- I will work entirely alone on this test, and all the solutions I submit will be my own work;
- I will not share my solutions to the test with anyone;
- I will not use a false identity, or take this test in somebody else's name;
- I will not engage in any unfair activities that will dishonestly improve my results or influence somebody else's results.

Python Essentials - Part 2 Summary Test

Time limit: 45 minutes
Number of questions: 30
Points to score: 30
Passing score: 70%

Start

The following line of code:

```
for line in open('text.txt', '
```

- (●) is valid as `open` returns an iterable object
- ( ) is invalid as `open` returns nothing
- ( ) may be valid if `line` is a list
- ( ) is invalid as `open` returns a non-iterable object

The following statement:

```
assert var != 0
```

- ( ) is erroneous
- (●) will stop the program when `var != 0`
- ( ) has no effect
- ( ) will stop the program when `var == 0`

What will be the output of the following code, located in the `p.py` file?

```
print(__name__)
```

- `__p.py__`
- **`__main__`**
- `main`
- `p.py`

The following code prints:

```
print(chr(ord('p') + 2))
```

- `q`
- **`r`**
- `t`
- `s`

What will be the effect of running the following code?

```
class A:
    def __init__(self, v):
        self.__a = v + 1

a = A(0)
print(a.__a)
```

- (•) it will raise an `AttributeError` exception
- ( ) it will print `0`
- ( ) it will print `1`
- ( ) it will print `2`

The following statement:

```
from a.b import c
```

causes the import of:

- ( ) entity `c` from module `b` from package `a`
- ( ) entity `b` from module `a` from package `c`
- (•) entity `c` from module `a` from package `b`
- ( ) entity `a` from module `b` from package `c`

What will be the output of the following code?

```
class A:
    def __init__(self, v=2):
        self.v = v

    def set(self, v=1):
        self.v += v
        return self.v

a = A()
b = a
b.set()
print(a.v)
```

- (●) `3`
- ( ) `1`
- ( ) `0`
- ( ) `2`

What will be the result of executing the following code?

```
class A:
    pass

class B(A):
    pass

class C(B):
    pass

print(issubclass(A, C))
```

- ( ) it will print `1`
- (●) it will print `False`
- ( ) it will raise an exception
- ( ) it will print `True`

What will be the result of executing the following code?

```
class A:
    def a(self):
        print('a')

class B:
    def a(self):
        print('b')


class C(B, A):
    def c(self):
        self.a()

o = C()
o.c()
```

- it will print `a`
- (•) it will print `b`
- it will print `c`
- it will raise an exception

← Prev Next →

What will be the result of executing the following code?

```
try:
    raise Exception(1, 2, 3)
except Exception as e:
    print(len(e.args))
```

- it will raise an unhandled exception
- it will print `2`
- (•) it will print `3`
- it will print `1`

What will be the result of executing the following code?

```
class I:
    def __init__(self):
        self.s = 'abc'
        self.i = 0
    def __iter__(self):
        return self
    def __next__(self):
        if self.i == len(self.s):
                raise StopIteration
        v = self.s[self.i]
        self.i += 1
        return v

for x in I():
    print(x, end='')
```

- (●) it will print `abc`
- ( ) it will print `210`
- ( ) it will print `012`
- ( ) it will print `cba`

← Prev Next →

The `sys.stderr` stream is normally associated with:

- ( ) the printer
- (●) the screen
- ( ) a null device
- ( ) the keyboard

What output will appear after running the following snippet?

```
import math
print(dir(math))
```

- a string containing the fully qualified name of the module
- **a list of all the entities residing in the `math` module**
- an error message
- the number of all the entities residing in the `math` module

The following code:

```
x = "\\\\"
print(len(x))
```

- will cause an error
- prints `1`
- prints `3`
- **prints `2`**

← Prev Next →

Assuming that the `open()` invocation has gone successfully, the following snippet will:

```
for x in open('file', 'rt'):
    print(x)
```

- read the file character by character
- cause an exception
- (•) read the file line by line
- read the whole file at once

← Prev | Next →

The following code:

```
x = "\\\"
print(len(x))
```

- (•) will cause an error
- prints `2`
- prints `1`
- prints `3`

← Prev | Next →

The following code:

```
print(float("1.3"))
```

- prints `1,3`
- prints `13`
- raises a `ValueError` exception
- prints `1.3`

What will be the output of the following code?

```
class A:
    A = 1
    def __init__(self):
        self.a = 0

print(hasattr(A, 'a'))
```

- `0`
- `False`
- `1`
- `True`

What will be the result of executing the following code?

```
class A:
    def __init__(self):
        pass

a = A(1)
print(hasattr(a, 'A'))
```

- (•) it will raise an exception
- ( ) it will print `True`
- ( ) it will print `False`
- ( ) it will print `1`

If `s` is a stream opened in *read* mode, the following line:

```
q = s.read(1)
```

will read:

- (•) one character from the stream
- ( ) one buffer from the stream
- ( ) one kilobyte from the stream
- ( ) one line from the stream

← Prev Next →

The compiled Python bytecode is stored in files which have their names ending with:

- [x] `py`
- [ ] `pyb`
- [ ] `pyc`
- [ ] `pc`

If you want to fill a byte array with data read in from a stream, you can use:

- [ ] the `read()` method
- [ ] the `readbytes()` method
- [ ] the `readfrom()` method
- [x] the `readinto()` method

If the class's constructor is declared as below, which one of the assignments is invalid?

```
class Class:
    def __init__(self, val=0):
        pass
```

- [ ] `object = Class(1)`
- [ ] `object = Class()`
- [ ] `object = Class(None)`
- [x] `object = Class(1, 2)`

Knowing that a function named `fun()` resides in a module named `mod`, and was imported using the following statement:

```
from mod import fun
```

Choose the right way to invoke the `fun()` function:

- `mod:fun()`
- `mod.fun()`
- [x] `fun()`
- `mod::fun()`

What will be the output of the following snippet?

```
try:
    raise Exception
except:
    print("c")
except BaseException:
    print("a")
except Exception:
    print("b")
```

- `a`
- `b`
- `1`
- [x] it will cause an error

If there are more than one `except:` branches after the `try:` clause, we can say that:

- not more than one `except:` block will be executed
- none of the `except:` blocks will be executed
- exactly one of the `except:` blocks will be executed
- one or more of the `except:` blocks will be executed

Assuming that the following three files: `a.py`, `b.py`, and `c.py` reside in the same folder, what will be the output produced after running the `c.py` file?

```
# file a.py
print("a", end='')

#file b.py
import a
print("b", end='')

#file c.py
print("c", end='')
import a
import b
```

- `bac`
- `cab`
- `abc`
- `cba`

← Prev Next →

What will be the result of executing the following code?

```
def I(n):
    s = '+'
    for i in range(n):
        s += s
        yield s

for x in I(2):
    print(x, end='')
```

- it will print `++`
- it will print `+++`
- **it will print `++++++`**
- it will print `+`

What will be the output of the following snippet?

```
try:
    raise Exception
except BaseException:
    print("a")
except Exception:
    print("b")
except:
    print("c")
```

- **`a`**
- `b`
- `1`
- it will cause an error

What will be the result of executing the following code?

```
def o(p):
    def q():
        return '*' * p
    return q

r = o(1)
s = o(2)
print(r() + s())
```

- (●) it will print `***`
- ( ) it will print `*`
- ( ) it will print `**`
- ( ) it will print `****`