Examination in *Programming in Python* BB1000

Suggested solutions

2018-08-15 08:00-13:00

Grading:

- E: Part 1 >= 75%
- D: Part 1 >= 75% and Part 2 >= 25%
- C: Part 1 >= 75% and Part 2 >= 75%
- B: Part 1 >= 75% and Part 2 >= 75% and Part 3 >= 25%
- A: Part 1 >= 75% and Part 2 >= 75% and Part 3 >= 75%

Each correctly answered question yields one point.

Note: Part 2 will only be graded if Part 1 has been passed. Part 3 will only be graded if both parts 1 and 2 both has been passed.

Part 1

1. A literal is a notation specific for a language to represent an object of a given datatype and value. Which of the following are true

   - `1` represents an object of type `int`, an integer with the numerical value one.
   - `1.0` represents an object of type `float`, an floating point number with the numerical value one.
   - `"1.0"` represents an object of type `str`, a string with three characters
   - `[]` represents an object of type `list`, an empty list
   - `()` represents an object of type `tuple`, an empty tuple
   - `{}` represents an object of type `dict`, an empty dictionary

   Answer: all are true

2. Reorder the lines (write numbers 1-4 next to the lines) to return the output 'data'

```
>>> a = 'd'
>>> a += 'a'                #<-----
>>> d = a + 't' + 'a'       #<-----
>>> print(d)
data
```

   Answer: 1, 3, 2, 4

3. Select the correct output line

```
>>> print(10 / 2, 10 % 2, 10 ** 2)
5.0 0 100

5 0 100
5 0 5
```

4. Which are valid ways to define a list

```
l = [2 4 6]             # a
l = ('a', 'b', 'c')     # b
l = [2, 4, 6]           # c  ✓
l = ['a', 'b', 'c']     # d  ✓
l = (2, 4, 6)           # e
l = '[2, 4, 6]'         # f
l = ['2, 4, 6']         # g  ✓
l = [2, '4', 6]         # h  ✓
```

5. What is the output of

```
>>> s = 'hello'
>>> print(s.index('l'))
2
```

6. Which of the following are class method calls.

```
[1, 2, 3].sort()   #a
print('horse')     #b
'horse'.upper()    #c
type([1, 2, 3])    #d
```

Answer: a, c

7. The type of an object determines the methods you can use with it. Select the correct output line for

```
l = ['a', 'b', 'c']
print(l.upper())

['a', 'b', 'c']                                          #a
['A', 'B', 'C']                                          #b
AttributeError: 'list' object has no attribute 'upper'   #c
```

Answer: c

8. Join the commands and explanation with lines

command	----- **explanation**
`git clone`	Get a copy of an existing repository
`git init`	Create a new repository
`git add`	Save changes in the work directory to the cache/td>
`git commit`	Save changes cache to the local repository

Part 2

9. What is the output from the last line

```
>>> l = list()
>>> l_backup = l
>>> l.append(1)
>>> print(l_backup)
[1]
```

Answer: `[1]`, the assignment assigns `l_backup` to the same object as `l`. Any change involving one variable will be reflected in the other

10. From the documentation of the built-in `round` function:

```
$pydoc round

round(...)
    round(number[, ndigits]) -> number

    Round a number to a given precision in decimal digits (default 0 digits).
    This returns an int when called with one argument, otherwise the
    same type as the number. ndigits may be negative.
```

What would the header of a valid function definition look like (first line)?

```
def round(number, ndigits=0):
    ...
```

Answer: `def round(number, ndigits=0):`

11. The built-in `sorted` function has the following documentation

```
$ pydoc sorted
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

What is the result of

```
>>> fruit = ['banana', 'grapefruit', 'lime', 'pineapple']
>>> print(sorted(fruit, key=len, reverse=False))
['lime', 'banana', 'pineapple', 'grapefruit']
```

12. Some function change objects "in place" while other return new objects. What it the output of

```
>>> s = 'horse'
>>> s.upper()
'HORSE'
>>> print(s)
horse
```

13. A Fibonacci sequence starts with 0, and 1, and all following elements constitute the sum of the two previous elements. Write the missing line for tuple packing and unpacking to generate the first ten numbers of the Fibonacci sequence

```
>>> fibonacci_numbers = []
>>> a, b = 0, 1
>>> for _ in range(10):
...     fibonacci_numbers.append(a)
...     a, b = b, a + b
>>> print(fibonacci_numbers)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

14. Outline a function `loadfile` which reads a file of numbers and returns a list of lists of floats (which represents a matrix.

```
>>> def loadfile(fileobj):
...     lines = []
...     for line in fileobj:
...          words = [float(word) for word in line.split()]
...          lines.append(words)
...     return lines
```

Expect the code to work like this: a test file `testfile.txt`

```
1 2
3 4
```

gives the following output

```
>>> with open('testfile.txt') as f:
...      mat = loadfile(f)
>>> print(mat)
[[1.0, 2.0], [3.0, 4.0]]
```

Use a combination of `float`, `split`, `append` functions/methods.

Note: the numpy library function `loadtxt` implements this behaviour but returns a two-dimensional numpy array

15. Update the script (fill in ____) to obtain the given output

```
>>> a = [3.69, 2.02, 7.92]
>>> b = ['z', 'a', 'r']
>>> c = [9, 7, 2]
>>> d = [a, b, c] #___

>>> print(d)
[[3.69, 2.02, 7.92], ['z', 'a', 'r'], [9, 7, 2]]
```

16. Update the script

```
>>> import numpy as np
>>> x = np.array(["Jan", "Feb", "Mar", "Apr", "May", "Jun"])
>>> print(x[1: 4])
['Feb' 'Mar' 'Apr']
```

Part 3

17. What is the output of

```
>>> def f(x, y, *args, a=1, b=2, **kwargs):
...     print('x', x)
...     print('y', y)
...     print('a', a)
...     print('b', b)
...     print('args', args)
...     print('kwargs', kwargs)


>>> f(1, 2, 3, a=4, b=5, c=6)
x 1
y 2
a 4
b 5
args (3,)
kwargs {'c': 6}
```

18. What is the output of

```
>>> def deco(f):
...     def inner():
...         return 'inner result'
...     return inner
>>> @deco
... def target():
...     return 'original result'
>>> print(target())
inner result
```

19. One of the application areas for decorators can be used to keep a register of functions. What is the output from

```
>>> def main():
...     print('<1>')
...     registry = []
...
...     def register(func):
...         print('<2>')
...         registry.append(func)
...         return func
...
...     @register
...     def f1():
...         print('<3>')
...     print('<4>')
...     f1()
>>> main()
<1>
<2>
<4>
<3>
```

20. Consider the following `cache` decorator.

```
>>> def cache(func):
...     def wrap(*args):
...         if args not in wrap.results:
...             wrap.results[args] = func(*args)
...         return wrap.results[args]
...     wrap.results = {}
...     return wrap

>>> @cache
... def slow_sum(x, y):
...     import time; time.sleep(5)
...     return x + y


>>> slow_sum(1, 1)
2
>>> slow_sum(1, 1)
2
```

Explain in plain language what effect the decorator has on a function, and in particular, what is the observable difference between the two function calls.

Answer: the results of the function call is saved in a dictionary so that when the function is called a second time with the same arguments the saved value is returned without any recalculation