Examination in *Programming in Python* BB1000

Suggested solutions

2019-08-16 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. Write a function definition that

   - takes one optional parameter with default value `None`
   - with no parameter it returns -999
   - the parameter can be assumed to be a sequence-like type (e.g. list, tuple)
   - the function returns the sum of the parameter elements if provided
   - an empty sequence gives zero

   Solution:

   ```
   >>> def sum_elements(seq=None):
   ...     if seq is None:
   ...         return -999
   ...     else:
   ...        return sum(seq)
   ```

   Such that

   ```
   >>> sum_elements()
   -999
   >>> sum_elements([])
   0
   >>> sum_elements((1, 2))
   3
   >>> sum_elements(range(4))
   6
   ```

2. List a way of looping over a dictionary (there are a few) so that e.g. `{"a": 1, "b": 2, "c": 3}` is displayed on the screen as

   ```
   a->1
   b->2
   c->3
   ```

   Solution:

   ```
   >>> d = {"a": 1, "b": 2, "c": 3}
   >>> for k, v in d.items():
   ...    print(f"{k}->{v}")
   a->1
   b->2
   c->3
   ```

3. A function takes a CSV-string with name/gender data as input and returns a dictionary with gender as key and a list of full names as value.

   e.g. with

   ```
   >>> sample_data = """first_name,last_name,gender
   ... Quintina,Firle,F
   ... Jesse,Nunson,M
   ... Lena,Stockley,F
   ... Margaux,Sirr,F
   ... Taylor,Alishoner,M"""
   ```

   Reorder and indent correctly the source lines:

   ```
   names = {"F": [], "M": []}
   return names
   def group_by_gender(data):
   first, last, gender = line.split(",")
   lines = data.split("\n")
   names[gender].append(f"{first} {last}")
   for line in lines[1:]:
   ```

   Solution:

   ```
   >>> def group_by_gender(data):
   ...     names = {"F": [], "M": []}
   ...     lines = data.split("\n")
   ...     for line in lines[1:]:
   ...         first, last, gender = line.split(",")
   ...         names[gender].append(f"{first} {last}")
   ...     return names
   ```

   such that

   ```
   >>> group_by_gender(sample_data)
   {'F': ['Quintina Firle', 'Lena Stockley', 'Margaux Sirr'], 'M': ['Jesse Nunson', 'Taylor Alishoner']}
   ```

4. A single line with `split` and `join` can split a multiline string

   ```
   >>> text = """one
   ... two
   ... three"""
   ```

   into

   ```
   'one|two|three'
   ```

   Write down an instruction with one or more lines that does this

   Solution:

   ```
   >>> text = """one
   ... two
   ... three"""
   >>> '|'.join(text.split('\n'))
   'one|two|three'
   ```

5. Outline a function which takes a string as input and returns a tuple with two objects

   - a modified string with vowels replaced with a star `'*'`
   - the number of replacements

   Solution:

   ```
   >>> def strip_vowels(text):
   ...     new = []
   ...     count = 0
   ...     for c in text:
   ...         if c.lower() in 'eiyaou':
   ...             new.append('*')
   ...             count += 1
   ...         else:
   ...             new.append(c)
   ...     modified = ''.join(new)
   ...     return (modified, count)
   ```

   such that

   ```
   >>> strip_vowels('hello world')
   ('h*ll* w*rld', 3)
   ```

6. What is the role of the binary operators `//` and `%` in Python arithmetics?

   Truncated integer division and modulus (remainder)

7. When calling a Python script from the command line

   ```
   $ python script.py arg1 arg2
   ```

   The arguments are saved in a specific data structure of a specific module What is the type and name of the data structure and which module?

   They are colleced in the list `sys.argv`

8. Give an example of an important usage of the variable `__name__` in Python scripts

   To filter out code that should not run during import of a module, that only runs when the script is the main program

   ```
   if __name__ == "__main__":
       ...
   ```

Part 2

9. Outline a class definition `Car` for a car with attributes make, model, year.

   Solution

   ```
   >>> class Car:
   ...     def __init__(self, make, model, year):
   ...         self.make = make
   ...         self.model = model
   ...         self.year = year
   ```

10. (Car continued)

    Add a class method so that default string representation of an object mimics the command for creation.

    Solution

    ```
    >>> class Car:
    ...     def __init__(self, make, model, year):
    ...         self.make = make
    ...         self.model = model
    ...         self.year = year
    ...
    ...     def __repr__(self):
    ...         return f"Car('{self.make}', '{self.model}', {self.year})"

    >>> car = Car('Volvo', 'Amazon', 1964)
    >>> repr(car)
    "Car('Volvo', 'Amazon', 1964)"
    ```

11. (Car continued)

    Add another class method so that the other string representation of an object reads

    Solution

    ```
    >>> class Car:
    ...     def __init__(self, make, model, year):
    ...         self.make = make
    ...         self.model = model
    ...         self.year = year
    ...
    ...     def __repr__(self):
    ...         return f"{self.__class__.__name__}('{self.make}', '{self.model}', {self.year})"
    ...
    ...     def __str__(self):
    ...         return f"{self.make} {self.model} ({self.year})"

    >>> car = Car('Volvo', 'Amazon', 1964)
    >>> str(car)
    'Volvo Amazon (1964)'
    ```

12. (Car Continued)

    Outline code that generate a list of Car objects by reading the a file `cars.csv` containing

```
Dodge,Charger,1969
GMC,Vandura G2500,1995
Toyota,Sienna,2007
Dodge,Challenger,2012
Pontiac,Grand Am,1989
Nissan,Altima,2009
Mazda,MPV,2002
Cadillac,DeVille,1994
Mercury,Tracer,1999
Volkswagen,Passat,1988
```

Solution:

```
>>> cars = []
>>> for line in open('cars.csv'):
...     make, model, year = line.strip().split(',')
...     car = Car(make, model, year)
...     cars.append(car)
```

such that

```
>>> cars
[Car('Dodge', 'Charger', 1969), Car('GMC', 'Vandura G2500', 1995), Car('Toyota', 'Sienna', 2007), Car('Dodge', 'Challenger', 2012), Car(
```

13. (Car Continued)

    The `sorted` function has the documentation

    ```
    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.
    ```

    How can you use this to sort the cars by year from newest to oldest?

    Solution:

    ```
    >>> def get_year(car):
    ...     return car.year

    >>> sorted(cars, key=get_year, reverse=True)
    [Car('Dodge', 'Challenger', 2012), Car('Nissan', 'Altima', 2009), Car('Toyota', 'Sienna', 2007), Car('Mazda', 'MPV', 2002), Car('Mer
    ```

14. (Car Continued)

    A car salesman wants to use your code but update to have a price attribute. Define a new class that inherits from Car with an initial price zero

    Solution:

    ```
    >>> class Car4Sale(Car):
    ...     def __init__(self, make, model, year, price=0):
    ...         super().__init__(make, model, year)
    ...         self.price = price
    ```

    such that

    ```
    >>> car = Car4Sale('Volvo', 'Amazon', 1964)
    >>> car.price
    0
    >>> car = Car4Sale('Volvo', 'Amazon', 1964, 9900)
    >>> car.price
    9900
    ```

15. (Car Continued)

    Write the function to calculate the total price for a list of cars

    Solution:

    ```
    >>> def sum_values(cars):
    ...     return sum(car.price for car in cars)

    >>> sum_values([])
    0
    >>> sum_values([
    ... Car4Sale('Chevrolet' ,'Silverado 3500',2003,34452),
    ... Car4Sale('Mazda', 626, 1991, 17121),
    ... Car4Sale('Oldsmobile', 'Achieva', 1993, 12982),
    ... ])
    64555
    ```

16. (Car Continued)

    When it comes to extending a class an alternative to inheritance is so called composition, which means in this case that a car and its price are separate data attributes of a new class

    ```
    >>> class CarWithPrice:
    ...     def __init__(self, car, price=0):
    ...         self.car = car
    ...         self.price = price
    ...
    ...     def __str__(self):
    ...         return f"{self.car}: {self.price}"
    >>> car = Car('Mercury', 'Sable', 1988)
    >>> car_price = CarWithPrice(car, 7000)
    ```

    What will be the output of `print(car_price)`?

    Solution:

    ```
    >>> print(car_price)
    ```

```
Mercury Sable (1988): 7000
```

Part 3

17. The documentation for `functools.partial` contains the following

    The partial() is used for partial function application which "freezes" some portion of a function's arguments and/or keywords resulting in a new object with a simplified signature. For example, partial() can be used to create a callable that behaves like the int() function where the base argument defaults to two:

    ```
    >>> from functools import partial
    >>> basetwo = partial(int, base=2)
    >>> basetwo.__doc__ = 'Convert base 2 string to an int.'
    >>> basetwo('10010')
    18
    ```

    Make an analogy of this for the print function such that objects are printed on separate lines.

    Solution:

    ```
        >>> lprint = partial(print, sep='\n')

    >>> lprint('a', 'b', 'c')
    a
    b
    c
    ```

    Hint: use the `sep` keyword argument

18. The zip documentation contains

    class zip(object) zip(iter1 [,iter2 [...]]) --> zip object

    *Return a zip object whose **next**() method returns a tuple where the i-th element comes from the i-th iterable argument. The **next**() method continues until the shortest iterable in the argument sequence is exhausted and then it raises StopIteration.*

    From this description what would be the output of

    ```
    l1 = [1, 2, 3]
    l2 = [4, 5]
    for z in zip(l1, l2):
        print(z)
    ```

    Solution:

    ```
        >>> l1 = [1, 2, 3]
        >>> l2 = [4, 5]
        >>> for z in zip(l1, l2):
        ...     print(z)
        (1, 4)
        (2, 5)
    ```

19. The map function has the following documentation

    class map(object) map(func, iterables) --> map object*

    Make an iterator that computes the function using arguments from each of the iterables. Stops when the shortest iterable is exhausted.

    What is the output of the following?

    ```
    l1 = [1, 2, 3]
    l2 = [4, 5]
    def f(x, y): return x + y
    for s in map(f, l1, l2):
        print(s)
    ```

    Solution:

    ```
        >>> l1 = [1, 2, 3]
        >>> l2 = [4, 5]
        >>> def f(x, y): return x + y
        >>> for s in map(f, l1, l2):
        ...     print(s)
        5
        7
    ```

20. The colorama module in Python can be used to give color output in a terminal e.g.

    ```
    >>> from colorama import Fore, Style
    >>> print(Fore.RED + 'some red text' + Style.RESET_ALL)
    ```

    ```
    some red text
    ```

    Use this to design a decorator such that all output from a decorated function is in red

    Solution:

    ```
        >>> def red(f):
        ...     def wrap(*args, **kwargs):
        ...         print(Fore.RED, end='')
        ...         f(*args, **kwargs)
        ...         print(Style.RESET_ALL)
        ...     return wrap
    ```

    such that

    ```
    >>> @red
    ... def hello():
    ```

```
...     print("Hello world!")
>>> hello()

Hello world!
```