

OJT - Python Exercise - Paper3

1. Write a program in which two strings are given and determine if they share a common substring. A substring may be as small as one character. The function returns either "YES" or "NO".
2. Write a decorator function that will record the number of times a function is called. Your decorator function should be called `record_calls` and `call_count` attribute that keeps track of the number of times it was called.
3. Write a function called `interleave` which accepts two iterables of any type and returns a new iterable with each of the given items "interleaved" (item 0 from iterable 1, then item 0 from iterable 2, then item 1 from iterable 1, and so on). An assumption here that both iterables contain the same number of elements.
4. Write `to_celsius` function that accepts a temperature in Fahrenheit as input and returns a temperature in Celsius.
5. Write a function that accepts an `iterable` and returns a new iterable with all items from the original iterable except for duplicates.
Ex. `uniques_only([1, 2, 2, 1, 1, 3, 2, 1])`
`[1, 2, 3]`
6. Write a function `to_percent` which accepts a number representing a ratio and returns a string representing the percentage representation of the number to one decimal place.
7. Write a function that accepts two strings and returns `True` if the two strings are `anagrams` of each other.
8. Write `Row` class that accepts any keyword arguments given to it and stores these arguments as attributes.

```
Ex. >>> row = Row(a=1, b=2)
>>> row.a
1
>>> row.b
2
```

9. Create a function `is_leap_year` that accepts a year and returns `True` if (and only if) the given year is a leap year.
10. Write a function `combine_lists` should take two lists and return a new list containing all elements from both lists.

11. Write a function, `last_lines`, which returns lines in a given ASCII text file in reverse order.

For example, given the following file, `my_file.txt`:

```
This is a file
This is line 2
And this is line 3
```

The `last_lines` function should work like this:

```
>>> for line in last_lines('my_file.txt'):
...     print(line, end=")
...
And this is line 3
This is line 2
This is a file
```

12. Write a function called `parse_ranges`, which accepts a string containing ranges of numbers and returns an iterable of those numbers.

Ex: `>>> parse_ranges('1-2,4-4,8-13')`
`[1, 2, 4, 8, 9, 10, 11, 12, 13]`

13. Write a function that accepts a string containing lines of numbers and returns a list of lists of numbers.

Ex. `matrix_from_string("3 4 5")`
`[[3.0, 4.0, 5.0]]`

14. Write a command-line program which helps a traveler keep track of the restaurants they've visited in different cities and what they thought of each. The program will accept two CSV files of restaurants, a "primary list" CSV and a "sublist" one, and update the primary one with new restaurants from the trip one.

15. Write a function `get_hypotenuse` that returns the hypotenuse of a right triangle given the other two sides.

```
>>> get_hypotenuse(0, 0)
0.0
```

```
>>> get_hypotenuse(3, 4)
5.0
```

16. Write a function `split_in_half` that splits a list in half and returns both halves.

```
>>> split_in_half([1, 2, 3, 4])
([1, 2], [3, 4])
```

17. Write a function that takes a sequence (like a list, string, or tuple) and a number `n` and returns the last `n` elements from the given sequence, as a list. For example:

```
>>> tail([1, 2, 3, 4, 5], 3)

[3, 4, 5]
```

18. Create your own exception.

19. Write a function that takes two strings representing dates and returns the string that represents the earliest point in time ? Ex. `get_earliest("01/27/1832", "01/27/1756")` return '01/27/1756'.

20. Create a function that determines which day of the month the San Diego Python meetup should be. It should accept year and month arguments and should return a `datetime.date` object representing the day of the month for the meetup.

```
>>> meetup_date(2012, 3)
datetime.date(2012, 3, 22)
```

21. Write a callable called `float_range` that acts sort of like the built-in `range` callable but it should allow for floating point numbers to be specified as start, stop, and step values.

```
>>> r = float_range(0.5, 2.5, 0.5)
>>> r
float_range(0.5, 2.5, 0.5)
>>> list(r)
[0.5, 1.0, 1.5, 2.0]
>>> len(r)
4
>>> for n in r:
...     print(n)
...
0.5
1.0
1.5
2.0
```

22. Write a function `is_iterator` so that it accepts an `iterable` and returns `True` if the given iterable is an `iterator`.

```
is_iterator(iter([]))
True
>>> is_iterator([1, 2])
False
```

23. Create a context manager. Context managers use a `with` block to bookend a block of code with automatic setup and tear down steps. Your context manager, `suppress`, should suppress exceptions of a given type:

```
>>> with suppress(NameError):
...     print("Hi!")
...     print("It's nice to meet you,", name)
...     print("Goodbye!")
...
Hi!
```

But exceptions of *other* types shouldn't be suppressed (we're suppressing a `TypeError` and a `NameError` is raised):

```
>>> with suppress(TypeError):
...     print("Hi!")
...     print("It's nice to meet you,", name)
...     print("Goodbye!")
...
Hi!
```

Traceback (most recent call last):

```
File "<stdin>", line 3, in <module>
```

```
NameError: name 'name' is not defined
```

24. Write a class that represents a circle. The circle should have a **radius**, a **diameter**, and an **area**. It should also have a nice string representation.
25. Write a program to convert integers to Roman numbers.
26. Write a function so that it accepts an iterable and returns True if the given iterable is an iterator.
27. Write a class that represents a bank account, do bank operations.
28. Standardize mobile numbers when given N mobile numbers. Sort them in ascending order. Print them in the standard format.
29. Write a function called `interleave` which accepts two iterables of any type and returns a new iterable with each of the given items "interleaved" (item 0 from iterable 1, then item 0 from iterable 2, then item 1 from iterable 1, and so on).
30. Convert each list element to a key-value pair.
ex:
Input : `test_list = [2323, 82, 129388, 95]`
Output : `{23: 23, 8: 2, 129: 388, 9: 5}`