Python exercise

Python 2 versun Python 3

- http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html
- http://py3readiness.org/

Editor

- http://www.sublimetext.com/

Practice

- Basic Practice
- More Mathematical (and Harder) Practice
- List of Practice Problems
- A SubReddit Devoted to Daily Practice Problems
- A very tricky website with very few hints and touch problems (Not for beginners but still interesting)

Dynamic typed

Reverse Index

- -1 is the last one
- trick: "String"[::-1])

Slice [Start:Stop:Size]

String

- Immutability
- letter*num
- upper(), lower(), split(), capitalize()
- join(list) -> '--'.join([1,2,3]) == '1--2--3'

.format()

- float: {value:width.precision f}

f-strings

- e.g. print(f'{name} is {age} years old')

Moreover

- https://pyformat.info/

List

- list = list1 + list2
- list.append()

- list.pop(index) -> return element
- list.sort() -> return None
- list.reverse() -> return None
- list*num -> [1]*2 == [1,1]

Dictionary

- may.keys()
- may.values()
- may.items()

Tuples

- Immutability
- tuples.count(key)
- tuples.index(key)

Set

- myset = set(list)
- myset.add()

I/O

%%writefile filename.txt
file context
myfile = open(filename.txt)

- **mode='r'** is read only
- **mode='w'** is write only (overwrite file or create new)
- **mode='a'** is append only (add on to file) -> myfile.write('new context')
- **mode='r+'** is reading and writting
- **mode='w+'** is writting and reading (overwrite file or create new)

myfile.close()

- with open(filename.txt) as myfile: myfile.read() -> no need use close()

myfile.read() -> return String
myfile.seek(0) -> to do myfile.read() again
myfile.readlines() -> return List with '\n' in the end

while ... else ...
break: break out of the current closest enclosing loop
continue: go to the top of the closest enclosing loop
pass: do nothing at all

Operators

- list(range(start, stop, stepSize)) returns list
- enumerate(list) returns (index, item)

- zip(list1, list2, list3, ...) returns tuples
- item in list/dictionary/dictionary.values()/dictionary.keys() returns boolean
- min(list)
- max(list)
- from random import shuffle -> shuffle(list) not returns anything
- from random import randint -> randint(begin, stop) returns a random num in the range
- result = input('input: ') returns any input as String
- string.capitalize() / string.title()

function

- e.g. def myfunc(value='default') -> set defualt value
- e.g. def myfunc(*arg) -> takes in an arbitrary number of argument as tuple
- e.g. def myfunc(**kwargs) -> accept arguments(as many as wanted) as dictionary

map/filter/lambda expression

- map(myfunc, list) returns myfunc(x) for x in list
- filter(myfunc, list) only returns myfunc(x) for x in list if condition is true
- map(lambda x: x/2, list) returns x/2 for x in list

LEGB rule

- L - local -> name assigned in any way within a function, and not declared global in that function
- E - Enclosing function locals -> names in the local scope of any and all enclosing functions, from inner to outer
- G - Global(module) -> names assigned at the top-level of a module file, or declared global in a def within the file
- B - Built-in(Python) -> names preassigned in the built-in names module: open, range, SyntaxError

```
# Globla
def function1():
    # Enclosing
    def function2 ():
        # Local
        # reassign global variable (better avoid using it)
        global x
```

Special (Magic/Dunder) methods

- **init**
- **str**
- **len**
- **del**

Module and Package

- **init**.py

- if **name** == "**main**"

Test tools

- pylint
- unittest

Decorators

```
def decorator_func(original_func):
        def wrap_func():
                # some extra code
                original_func()
                # some extra code
        return wrap_func

new_func = decorator_func(some_func)
```

```
@new_decorator
def some_func():
        pass
```

Python Web page Framework

- decorators
- Flask
- Django

Python Generator

- yield -> memory efficient
- iter()

Advanced modules

- from collections import Counter

```
sum(c.values())                 # total of all counts
c.clear()                       # reset all counts
list(c)                         # list unique elements
set(c)                          # convert to a set
dict(c)                         # convert to a regular dictionary
c.items()                       # convert to a list of (elem, cnt) pairs
Counter(dict(list_of_pairs))    # convert from a list of (elem, cnt) pairs
c.most_common()[:-n-1:-1]       # n least common elements
c += Counter()                  # remove zero and negative counts
```

- from collections import defaultdict -> return default dictionary
- from collections import OrderedDict -> return ordered dictionary
- from collections import namedtuple -> create a new object/class type with some attribute fields
- pdb -> debugger pdb.set_trace()
- timeit -> timeit.timeit(func, number=step_num) / %timeit func
- regular expressions

Escape Codes

You can use special escape codes to find specific types of patterns in your data, such as digits, non-digits, whitespace, and more. For example:

Code	Meaning
`\d`	a digit
`\D`	a non-digit
`\s`	whitespace (tab, space, newline, etc.)
`\S`	non-whitespace
`\w`	alphanumeric
`\W`	non-alphanumeric