**Python 2**

**Excercise Answers**

1. Demonstrate your understanding of a few "dunder methods" in Python

Dunder methods are names that are preceded and succeeded by double underscores, hence the name dunder. They are also called magic methods and can help override functionality for built-in functions for custom classes.

def \_\_init\_\_(self, names)names):

if names:

self.names = names.copy()

for name in names:

self.versions[name] = 1

else:

raise Exception("Please Enter the names")

The init method is used to create an instance of the class.

2. What is the difference between using "\_\_repr\_\_" and "\_\_str\_\_"? Explain with a code snippet

repr – goal is to be unambiguos

str – goal is to be readable

>>> from datetime import datetime

>>> a = datetime.now()

>>> str(a)

'2022-09-16 14:52:52.286501'

>>> repr(a)

'datetime.datetime(2022, 9, 16, 14, 52, 52, 286501)'

3. Implement and show the use of

* 1. \_\_getitem\_\_, \_\_setitem\_\_, \_\_delitem\_\_, \_\_contains\_\_
  2. \_\_len\_\_
  3. \_\_call\_\_

\_\_enter\_\_ , \_\_exit\_\_

a.

class software:

names = []

versions = {}

def \_\_init\_\_(self, names):

if names:

self.names = names.copy()

for name in names:

self.versions[name] = 1

else:

raise Exception("Enter names")

def \_\_str\_\_(self):

s ="The current softwares and their versions are listed below: \n"

for key,value in self.versions.items():

s+= f"{key} : v{value} \n"

return s

def \_\_setitem\_\_(self,name,version):

if name in self.versions:

self.versions[name] = version

else:

raise Exception("Software Name doesn't exist")

def \_\_getitem\_\_(self,name):

if name in self.versions:

return self.versions[name]

else:

raise Exception("Software Name doesn't exist")

def \_\_delitem\_\_(self,name):

if name in self.versions:

del self.versions[name]

self.names.remove(name)

else:

raise Exception("Software Name doesn't exist")

def \_\_len\_\_(self):

return len(self.names)

def \_\_contains\_\_(self,name):

if name in self.versions:

return True

else:

return False

p = software(['S1', 'S2', "S3"])

p['S1'] = 2

# p['2'] = 2

print(p['S2'])

del p['S1']

print(p)

if 'S2' in p:

print("Software Exists")

else:

print("Software DOESN'T exist")

**\_\_call\_\_** - this method is called when functions are called

**\_\_enter\_\_ and \_\_exit\_\_ -** also called as context manager special methods in python. You may come up with the question "what is a context manager ?". A context manager is a manager which has the ability to make avail the data withing the context or within a block of code and destroy it when it is used. We implement context manager functionality using with keyword in python programming.

# file resource management without using context managers

file = open('sample.txt', 'w')

try:

file.write('Python is super awesome!')

finally:

file.close()

# file resource management with using context managers

with open('sample.txt', 'w') as file:

file.write('Python is super awesome!')

internal implementation of the "open" keyword

import io

class open(object):

def \_\_init\_\_(self, file\_name, mode='r'):

print('\_\_init\_\_ called')

self.file\_name = file\_name

self.mode = mode

def \_\_enter\_\_(self):

print('\_\_enter\_\_ called')

self.file = io.open(self.file\_name, self.mode)

return self.file

def \_\_exit\_\_(self, exc\_type, exc\_value, traceback):

print('\_\_exit\_\_ called')

self.file.close()

# lets test it

with open('sample.txt', 'w') as f:

f.write('hello')

# output

# \_\_init\_\_ called

# \_\_enter\_\_ called

# \_\_exit\_\_ called

**\_\_contains\_\_** method defines how instances of class behave when they appear at right side of in and not in operator.

class Person(object):

def \_\_init\_\_(self,name,age):

self.name = name

self.age = age

def \_\_contains\_\_(self,param1):

return True if param1 in self.\_\_dict\_\_.keys() else False

>>> p = Person('Robby Krieger',23)

>>> 'name' in p

True

4. What are the ways to initialize a list? Show them by implementing short code snippet?

i) list = [1,2,3,”a”,”b”]

ii) using looping and append

iii) list comprehension

iv) using \* --> l = [0]\*10

5. List Vs Tuple

list is mutable and rounded with square bracket

tuple is immutable and rounded with paranthesis

l = [1,2,3,4]

t = (1,2,3,4)

l[2] = 5 is possible

t[2] = 5 not possible

6. Show how to sort a list of tuples by ordering based on the second element of the tuple. Feel free to use built-in functions. ex: input: [(1, 2), (3, 6), (5, 1)] => output: [(5, 1), (1, 2), (3, 6)]

l = [(1, 2), (3, 6), (5, 1)]

print(sorted(l, key = lambda x : x[1]))

7. What are the new "dunder" methods you found for the new Mapping related types such as dict and set?

\_\_len\_\_(self)

\_\_getitem\_\_(self, key)

\_\_setitem\_\_(self, key, value)

\_\_delitem\_\_(self, key)

\_\_iter\_\_(self)

\_\_reversed\_\_(self)

\_\_contains\_\_(self, item)

\_\_missing\_\_

The \_\_missing\_\_(self, key) method defines the behavior of a [dictionary](https://blog.finxter.com/python-dictionary/) [subclass](https://blog.finxter.com/inheritance-in-python-harry-potter-example/) if you access a non-existent key. More specifically, Python’s \_\_getitem\_\_() dictionary method internally calls the \_\_missing\_\_() method if the key doesn’t exist.

8. Show usage of defaultdict, missingitem and setdefault

**setdefault**

Here’s an example of how you can use .setdefault() to handle missing keys in a dictionary:

>>> a\_dict = {}

>>> a\_dict['missing\_key']

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

a\_dict['missing\_key']

KeyError: 'missing\_key'

>>> a\_dict.setdefault('missing\_key', 'default value')

'default value'

>>> a\_dict['missing\_key']

'default value'

>>> a\_dict.setdefault('missing\_key', 'another default value')

'default value'

>>> a\_dict

{'missing\_key': 'default value'}

In the above code, you use .setdefault() to generate a default value for missing\_key. Notice that your dictionary, a\_dict, now has a new key called missing\_key whose value is 'default value'. This key didn’t exist before you called .setdefault(). Finally, if you call .setdefault() on an existing key, then the call won’t have any effect on the dictionary. Your key will hold the original value instead of the new default value.

**\_\_missing\_\_()**

The following code shows a viable Python implementation for .\_\_missing\_\_():

def \_\_missing\_\_(self, key):

if self.default\_factory is None:

raise KeyError(key)

if key not in self:

self[key] = self.default\_factory()

return self[key]

**defaultdict**

Take a look at how you can create and properly initialize a defaultdict:

>>> # Correct instantiation

>>> def\_dict = defaultdict(list) # Pass list to .default\_factory

>>> def\_dict['one'] = 1 # Add a key-value pair

>>> def\_dict['missing'] # Access a missing key returns an empty list

[]

>>> def\_dict['another\_missing'].append(4) # Modify a missing key

>>> def\_dict

defaultdict(<class 'list'>, {'one': 1, 'missing': [], 'another\_missing': [4]})

**9. Counter and ordereddict**

**Counter**

Python dictionary-like class [collections.Counter](https://realpython.com/python-counter/) is specially designed for counting items in a sequence. With Counter, you can write the mississippi example as follows:

>>> from collections import Counter

>>> counter = Counter('mississippi')

>>> counter

Counter({'i': 4, 's': 4, 'p': 2, 'm': 1})

**OrderedDict**

>>> from collections import OrderedDict

>>> numbers = OrderedDict()

>>> numbers["one"] = 1

>>> numbers["two"] = 2

>>> numbers["three"] = 3

>>> numbers

OrderedDict([('one', 1), ('two', 2), ('three', 3)])