Exam #0

**Task 1.** Fill missing pieces (\_\_\_\_) of the following code such that prints make sense.

name = 'John Doe'

if \_\_\_\_:

print('Name "{}" is more than 20 chars long'.format(name))

length\_description = 'long'

elif \_\_\_\_:

print('Name "{}" is more than 15 chars long'.format(name))

length\_description = 'semi long'

elif \_\_\_\_:

print('Name "{}" is more than 10 chars long'.format(name))

length\_description = 'semi long'

elif \_\_\_\_:

print('Name "{}" is 8, 9 or 10 chars long'.format(name))

length\_description = 'semi short'

else:

print('Name "{}" is a short name'.format(name))

length\_description = 'short'

name = 'John Doe'

if len(name) > 20 :

print('Name "{}" is more than 20 chars long'.format(name))

length\_description = 'long'

elif len(name) > 15 :

print('Name "{}" is more than 15 chars long'.format(name))

length\_description = 'semi long'

elif len(name) > 10 :

print('Name "{}" is more than 10 chars long'.format(name))

length\_description = 'semi long'

elif len(name) == 8 or len(name) == 9 or len(name) == 10:

print('Name "{}" is 8, 9 or 10 chars long'.format(name))

length\_description = 'semi short'

else:

print('Name "{}" is a short name'.format(name))

length\_description = 'short'

**Task 2.** Fill the \_\_\_\_ parts in the code below.

words = ['PYTHON', 'JOHN', 'chEEse', 'hAm', 'DOE', '123']

upper\_case\_words = []

for \_\_\_\_ in words:

if \_\_\_\_.isupper():

\_\_\_\_.append(\_\_\_\_)

words = ['PYTHON', 'JOHN', 'chEEse', 'hAm', 'DOE', '123']

upper\_case\_words = []

for item in words:

if item.isupper():

upper\_case\_words.append(item)

for item in upper\_case\_words:

print(item)

**Task 3.** Fill \_\_\_\_ pieces of the count\_even\_numbers implemention in order to pass the assertions. You can assume that numbers argument is a list of integers.

\_\_\_\_ count\_even\_numbers(numbers):

count = 0

for num in \_\_\_\_:

if \_\_\_\_ % 2 == \_\_\_\_:

count += \_\_\_\_

\_\_\_\_\_ \_\_\_\_\_

assert count\_even\_numbers([1, 2, 3, 4, 5, 6]) == 3

assert count\_even\_numbers([1, 3, 5, 7]) == 0

assert count\_even\_numbers([-2, 2, -10, 8]) == 4

def count\_even\_numbers(numbers):

count = 0

for num in numbers:

if num % 2 == 0:

count += 1

return count

assert count\_even\_numbers([1, 2, 3, 4, 5, 6]) == 3

assert count\_even\_numbers([1, 3, 5, 7]) == 0

assert count\_even\_numbers([-2, 2, -10, 8]) == 4

**Task 4.** Create a list of strings based on a list of numbers[¶](https://nbviewer.jupyter.org/github/jerry-git/learn-python3/blob/master/notebooks/beginner/exercises/for_loops_exercise.ipynb#3.-Create-a-list-of-strings-based-on-a-list-of-numbers)

The rules:

If the number is a multiple of five and odd, the string should be 'five odd'

If the number is a multiple of five and even, the string should be 'five even'

If the number is odd, the string is 'odd'

If the number is even, the string is 'even'

numbers = [1, 2, 4, 6, 81, 80, 100, 95]

# Your implementation

my\_list =

assert my\_list == ['odd', 'even', 'even', 'even', 'odd', 'five even', 'five even', 'five odd']

numbers = [1, 2, 4, 6, 81, 80, 100, 95]

my\_list = []

for item in numbers:

if (item%5 == 0 and item%2 == 0):

my\_list.append('five even')

elif(item%5 ==0 and item%2 !=0):

my\_list.append('five odd')

elif(item%2==0):

my\_list.append('even')

else:

my\_list.append('odd')

assert my\_list == ['odd', 'even', 'even', 'even', 'odd', 'five even', 'five even', 'five odd']

**Task 5**. Visualize data

a) Create plot of two sin function with distance of pi

import matplotlib.pyplot as plt

import numpy as np

%matplotlib inline

x = np.arange(0,4\*np.pi,0.1)

y = np.sin(x)

z = np.sin(y)

plt.plot(x,y,x,z)

plt.show()

b) Create pie chart of data pieChartData = [12,1,34]

import matplotlib.pyplot as plt

%matplotlib inline

pieChartData = (12, 1, 34)

plt.pie(pieChartData)

plt.show()

c) Create histogram for data with use of np.random.normal

import matplotlib.pyplot as plt

import numpy as np

mu, sigma = 0.5, 0.1

s = np.random.normal(mu, sigma, 1000)

count, bins, ignored = plt.hist(s, 20, normed=True)

plt.show()

d) Create boxplot for two columns with use of np.random.normal

import numpy as np

import matplotlib as mpl

mpl.use('agg')

import matplotlib.pyplot as plt

np.random.seed(10)

collectn\_1 = np.random.normal(100, 10, 200)

collectn\_2 = np.random.normal(80, 30, 200)

data\_to\_plot = [collectn\_1, collectn\_2]

fig = plt.figure(1, figsize=(9, 6))

ax = fig.add\_subplot(111)

bp = ax.boxplot(data\_to\_plot)

fig.savefig('fig1.png', bbox\_inches='tight')

e) Plot previous graphs next to each other

import numpy as np

import matplotlib as mpl

import matplotlib.pyplot as plt

mpl.use('agg')

import matplotlib.pyplot as plt

plt.subplot(1, 4, 1)

np.random.seed(10)

collectn\_1 = np.random.normal(100, 10, 200)

collectn\_2 = np.random.normal(80, 30, 200)

data\_to\_plot = [collectn\_1, collectn\_2]

fig = plt.figure(1, figsize=(9, 6))

ax = fig.add\_subplot(111)

bp = ax.boxplot(data\_to\_plot)

fig.savefig('fig1.png', bbox\_inches='tight')

plt.subplot(1, 4, 2)

mu, sigma = 0.5, 0.1

s = np.random.normal(mu, sigma, 1000)

count, bins, ignored = plt.hist(s, 20, normed=True)

plt.show()

plt.subplot(1, 4, 3)

pieChartData = (12, 1, 34)

plt.pie(pieChartData)

plt.show()

x = np.arange(0,4\*np.pi,0.1)

y = np.sin(x)

z = np.sin(y)

plt.subplot(1, 4, 4)

plt.plot(x,y,x,z)

plt.show()

**Task 7.** With use of Linear Models for Regression & Classification¶ from Lab #7 generate Simple Linear Regression using Ordinary Least Squares for

n\_features=1, noise=10, n\_samples=10203

As a result provide

* coef and intercept value

Paste your solution here.

* and graph of predicted vs. trained regression

Paste your solution here.