PythonNotebook2_solution_2023

November 27, 2023

Exercise 2.1.1

Calculate the area of a circle with a radius of 4 cm in the box below using the calculate_circle_area function.

[ ]:
```
r = ...
circle_area = ...

###BEGIN SOLUTION TEMPLATE=
r = 4
circle_area = calculate_circle_area(r)
###END SOLUTION

print(circle_area)
```

Exercise 2.1.2 Add a description to the calculate_circle_area(r) function below, as a docstring.

[ ]:
```
def calculate_circle_area(r):
    '''...this function calculates the area of a circle...!'''
    pi_circle = 3.141592653589793
    area = pi_circle*(r**2)
    return area
```

(Fixing) Exercise 2.1.3

Below is a function that should calculate the water pressure in the sea, as a function of depth. But it isn't working. Can you spot the error and fix it? Do not change any other line except the line with the error.

[ ]:
```
def water_pressure(z):
    '''Calculates the water pressure in the sea at a depth of input z (meters).␣
 ↪returns a value in Bar.'''

    water_density = 1000          #kg/m^3
g = 9.81                        #m/s^2
    p_atm = 100000                #Pa

    p_z = (p_atm + (z * water_density) * g) / 100000

    return p_z
```

```
###BEGIN SOLUTION TEMPLATE=
def water_pressure(z):
    '''Calculates the water pressure in the sea at a depth of input z (meters).␣
 ↪returns a value in Bar.'''

    water_density = 1000          #kg/m^3
    g = 9.81                      #m/s^2
    p_atm = 100000                #Pa

    p_z = (p_atm + (z * water_density) * g) / 100000

    return p_z

###END SOLUTION

calculated_pressure = water_pressure(2000)
print(calculated_pressure)
```

Exercise 2.1.4

Write a function called k_e() to calculate the kinetic energy of some object. It should have mass and velocity as its arguments. In case you forgot, the kinetic energy equation is

$$E_k = \frac{mv^2}{2}$$

[ ]:
```
#write your funtion below
...
...

###BEGIN SOLUTION TEMPLATE=
def k_e(mass, velocity):
    return 0.5 * mass * velocity ** 2
###END SOLUTION
print(k_e(10, 5))
```

(Searching) Exercise 2.1.5

Use the print() and k_e() functions to print a nice formatted output for any arbitrary input. Example of a desired output: The kinetic energy is: 200 J

[ ]:
```
#write your code here
...
...

###BEGIN SOLUTION TEMPLATE=
# k_e(x,y) = 200 # <= correct is to find x,y so that the function returns 200.
myvar = 200 # <= lazy way
print(f"The kinectic energy is: {myvar} J.")
```

```
###END SOLUTION
```

(Fixing) Exercise 2.1.6 The function below does not give the right answer. Could you fix it?

[ ]:
```
def add_two(numb):
    """
    add_two(numb) function -> takes the input numb and adds 2 to it

    Input:
        numb -> an integer, to which 2 must be added

    Output:
        ret -> a return integer, which is equal to numb + 2
    """
    ret = add_one(add_one(add_one(numb)))
    return ret

def add_one(number):
    """
    add_one(number) function -> takes the input number and adds 1 to it

    Input:
        number -> an integer, to which 1 must be added

    Output:
        ret -> a return integer, which is equal to number + 1
    """
    ret = number
    return ret


###BEGIN SOLUTION TEMPLATE=
def add_two(numb):
    """
    add_two(numb) function -> takes the input numb and adds 2 to it

    Input:
        numb -> an integer, to which 2 must be added

    Output:
        ret -> a return integer, which is equal to numb + 2
    """
    ret = add_one(add_one(numb))
    return ret

def add_one(number):
    """
```

```
    add_one(number) function -> takes the input number and adds 1 to it

    Input:
        number -> an integer, to which 1 must be added

    Output:
        ret -> a return integer, which is equal to number + 1
    """
    ret = number + 1
    return ret
###END SOLUTION

x = 5
y = add_two(x)

print(f'{x} + 2 is {y}, which is {y == x + 2}')
if y != x + 2:
    print('Something is wrong here...')
else:
    print('Looks gucci')
```

Exercise 2.3.1

One of the most crucial applications of if statements is filtering the data from errors and checking whether an error is within a certain limit.For example, checking whether the difference between an estimated value and the actual value are within a certain range. Mathematically speaking, this can be expressed as

$$|\hat{y} - y| < \epsilon$$

where $\hat{y}$ is your estimated value, y is the actual value and ϵ is a certain error threshold.The function check_error_size() below must do the same — it should return True if the error is within the acceptable range eps and False if it is not.

[ ]:
```
def check_error_size(estimated_value, true_value, eps):
    ... # your if statement (error in acceptable range)
        return True
    ... # statement if not in acceptable range
        return False

###BEGIN SOLUTION TEMPLATE=
def check_error_size(estimated_value, true_value, eps):
    if abs(estimated_value - true_value) <= eps:
        return True
    else:
        return False
###END SOLUTION

#You can try to check it by yourself by running the function with some␣
 ↪self-chosen numbers
```

```
check_error(...)
```

Exercise 2.3.2

Use the knowledge you have obtained in this Notebook and write your very own function to classify soil samples based on the average grain size. The classification you have to implement is the following:

1. Clay: avg grain size < 0.002 mm
2. Silt: 0.002 mm $\leq$ avg grain size < 0.063 mm
3. Sand: 0.063 mm $\leq$ avg grain size < 2 mm
4. Gravel: 2 mm $\leq$ avg grain size < 63 mm

Your task is to write the function, which will return the name of the soil type based on the provided average grain size in meters.

[ ]:
```
def classify_soil(avg_grain_size):
    ...

###BEGIN SOLUTION TEMPLATE=
def classify_soil(avg_grain_size):
    if avg_grain_size < 0.002:
        return 'Clay'
    elif avg_grain_size >= 0.002 and avg_grain_size < 0.063:
        return 'Silt'
    elif avg_grain_size >= 0.063 and avg_grain_size < 2:
        return 'Sand'
    elif avg_grain_size >= 2:
        return 'Gravel'
###END SOLUTION

print(classify_soil(1.5))
```

Exercise 2.3.3 — Triangle Inequality

Let's imagine that you received a request to manage a web-application, which provides and visualizes InSAR data of the estimated displacement in an area.The app is already quite cool, however, there is always room for improvement. For example, one might want to look into the statistics over the area of a triangle.Your goal is to help implement such functionality by checking whether the user selected a valid triangle. You need to do this by checking that the length of each of the sides is smaller than the sum of the other two.

[ ]:
```
def check_triangle(side_a, side_b, side_c):
    ...

###BEGIN SOLUTION TEMPLATE=
def check_triangle(side_a, side_b, side_c):
    part1 = side_a < (side_b + side_c)
    part2 = side_b < (side_a + side_c)
    part3 = side_c < (side_a + side_b)
```

```
    if part1 and part2 and part3:
        return "Valid Triangle"
    else:
        return "Invalid Triangle"
###END SOLUTION
```

Exercise 2.4.1

Now, let's get down to practice.Your first task is to finish the pack_variables() function — a function which will combine all inputs in one list and return it.More precisely, this function will receive 5 arguments as input and you have to return a list with these 5 elements inside.

[ ]:
```
def pack_variables(arg1, arg2, arg3, arg4, arg5):
    ...

###BEGIN SOLUTION TEMPLATE=
def pack_variables(arg1, arg2, arg3, arg4, arg5):
    pack = [arg1, arg2, arg3, arg4, arg5]
    return pack
###END SOLUTION

print(pack_variables(1, 2, 4, 22, 7))
```

Exercise 2.4.2

Here, you will have to perform a quality assessment on a received list of GNSS measurements. But a simple one.You have to check whether the received data has more than 1000 measurements, in that case we know the receiver was running for a long time without being interrupted. If it is shorter than 1000, we assume there were some interruptions. Hence, your function should return a message whether the data is fine or not.

[ ]:
```
def check_data(measurements):
    ...

###BEGIN SOLUTION TEMPLATE=
def check_data(measurements):
    if len(measurements) < 1000:
        return "Something wrong"
    else:
        return "All fine"
###END SOLUTION

#You can check your code below, where we simulate 1250 measurements
import random

gnss_data = [random.random() * 2.2e8 for i in range(1250)]
print(check_data(gnss_data))
```

Exercise 2.4.3

In this exercise you need to write something opposite to a packager… an unpacker! Sometimes it is easier to work with the most convenient data type, that is best suited for a specific task. In this example, you have to write a function which accepts data stored in a dictionary and saves some data from it. More precisely, you have to select x and y coordinates saved in the input_dict dictionary, and return them as a tuple, with x being the first entry.

[ ]:
```
# you do not have to change anything in this cell

input_dict = {
    'ID': '334856',
    'operator_name': 'Jarno',
    'observation_amount': '2485',
    'loc_x': [2, 5, 10, 12, -5, 8, 27, 1],
    'loc_y': [6, 1, -5, 15, 4, 8, 0, 10]
}
```

[ ]:
```
def unpack_dictionary(input_dict):
    ...

###BEGIN SOLUTION TEMPLATE=
def unpack_dictionary1(input_dict):

    ### OPTION 1: using `zip`, although not shown in the lecture
    loc_x = input_dict['loc_x']
    loc_y = input_dict['loc_y']

    return list(zip(loc_x, loc_y))

def unpack_dictionary2(input_dict):
    ### OPTION 2: using a for loop
    loc_x = input_dict['loc_x']
    loc_y = input_dict['loc_y']
    locations = []
    for i in range(len(loc_x)):
        locations.append((loc_x[i], loc_y[i]))
    return locations
###END SOLUTION

print(unpack_dictionary1(input_dict))
print(unpack_dictionary2(input_dict))
```