06 Exercises

September 13, 2020

0.1 Exercise 06.1 (selecting and passing data structures)

The task in Exercise 04 for computing the area of a triangle involved a function with six arguments (x and y components of each vertex). With six arguments, the likelihood of a user passing arguments in the wrong order is high.

Use an appropriate data structure, e.g. a `list`, `tuple`, `dict`, etc, to develop a new version of the function with a simpler interface (the interface is the arguments that are passed to the function). Add appropriate checks inside your function to validate the input data.

[0]:
```
#List
def area(x, y):
  triangle = abs((x[0]*(y[1]-y[2]))+(x[1]*(y[2]-y[0]))+(x[2]*(y[0]-y[1])))/2
  return triangle

x = [0.0, 0.0, 3.0]
y = [0.0, 2.0, 0.0]

print(type(x))

A = area(x, y)
print(A)
assert round(A - 3.0, 10) == 0.0
```

```
<class 'list'>
3.0
```

[0]:
```
#Tupple
def area(x, y):
  triangle = abs((x[0]*(y[1]-y[2]))+(x[1]*(y[2]-y[0]))+(x[2]*(y[0]-y[1])))/2
  return triangle

x = (0.0, 0.0, 3.0)
y = (0.0, 2.0, 0.0)

print(type(x))

B = area(x, y)
print(B)
```

```
assert round(B - 3.0, 10) == 0.0
```

```
<class 'tuple'>
3.0
```

[0]:
```
#Dictionary
def area(t):
  triangle =␣
 ↪abs((t['x0']*(t['y1']-t['y2']))+(t['x1']*(t['y2']-t['y0']))+(t['x2']*(t['y0']-t['y1'])))/
 ↪2
  return triangle

data={
    'x0':0.0, 'y0':0.0,
    'x1':0.0, 'y1':2.0,
    'x2':3.0, 'y2':0.0
}
print(type(data))

C = area(data)
print(C)
assert round(C - 3.0, 10) == 0.0
```

```
<class 'dict'>
3.0
```

0.2 Exercise 06.2 (selecting data structures)

For a simple (non-intersecting) polygon with n vertices, (x_0, y_0), (x_1, y_1), . . . , (x_{n-1}, y_{n-1}), the area A is given by

$$A = \left| \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) \right|$$

and where $(x_n, y_n) = (x_0, y_0)$. The vertices should be ordered as you move around the polygon.

Write a function that computes the area of a simple polygon with an arbitrary number of vertices. Test your function for some simple shapes. Pay careful attention to the range of any loops.

[0]:
```
```

[0]:
```
n = 2
p = [(17,12),(30,4),(105,67),(132,19)]

polygon=0
for i in range(n+1):
  polygon += (p[i][0]*p[i+1][1])-(p[i+1][0]*p[i][1])

A = abs(polygon/2)
print(A)
```

```
2775.5
```

0.3 Exercise 06.3 (indexing)

Write a function that uses list indexing to add two vectors of arbitrary length, and returns the new vector. Include a check that the vector sizes match, and print a warning message if there is a size mismatch. The more error information you provide, the easier it would be for someone using your function to debug their code.

Add some tests of your code.

Hint: You can create a list of zeros of length `n` by

```
z = [0]*n
```

Optional (advanced) Try writing a one-line version of this operation using list comprehension and the built-in function `zip`.

[0]:
```
def sum_vector(x, y):
    "Return sum of two vectors"
    z=[]
    if len(x) != len(y):
        print('vector size mismatch')
    else:
        for i in range(len(x)):
            z.append(x[i]+y[i])
    return z
```

[0]:
```
a = [0, 4.3, -5, 7]
b = [-2, 7, -15, 1]

c = sum_vector(a, b)

print(c)
assert c == [-2, 11.3, -20, 8]
```

```
[-2, 11.3, -20, 8]
```

0.3.1 Extension: list comprehension

[0]:
```
a = [0, 4.3, -5, 7]
b = [-2, 7, -15, 1]
c = [i + j for i, j in zip(a, b)]

print(c)
assert c == [-2, 11.3, -20, 8]
```

```
[-2, 11.3, -20, 8]
```

0.4 Exercise 06.4 (dictionaries)

Create a dictionary that maps college names (the key) to college abbreviations for at least 5 colleges (you can find abbreviations at https://en.wikipedia.org/wiki/Colleges_of_the_University_of_Cambridge#Colleges). From the dictionary, produce and print

1. A dictionary from college abbreviation to name; and
2. A list of college abbreviations sorted into alphabetical order.

Optional extension: Create a dictionary that maps college names (the key) to dictionaries of:

- College abbreviation
- Year of foundation
- Total number students

for at least 5 colleges. Take the data from https://en.wikipedia.org/wiki/Colleges_of_the_University_of_Cambric
Using this dictionary,

1. Find the college with the greatest number of students and print the abbreviation; and
2. Find the oldest college, and print the number of students and the abbreviation for this college.

[0]:
```
college = {
    'Clare':'CL',
    'Corpus Christi':'CC',
    'Churchill':'CHU',
    'Darwin':'DAR',
    'Clare Hall':'CLH'}
print(college)
# Create empty dictionary
college_inverse = {}
# Build inverse dictionary to map 'abreviation' -> name
for name, abbreviation in college.items():
    # Insert entry into dictionary
    college_inverse[abbreviation] = name

print(college_inverse)

#Sorting By Keys
print(sorted(college_inverse.keys()))
print(sorted(college_inverse.items()))
```

```
{'Clare': 'CL', 'Corpus Christi': 'CC', 'Churchill': 'CHU', 'Darwin': 'DAR',
'Clare Hall': 'CLH'}
{'CL': 'Clare', 'CC': 'Corpus Christi', 'CHU': 'Churchill', 'DAR': 'Darwin',
'CLH': 'Clare Hall'}
['CC', 'CHU', 'CL', 'CLH', 'DAR']
[('CC', 'Corpus Christi'), ('CHU', 'Churchill'), ('CL', 'Clare'), ('CLH', 'Clare
Hall'), ('DAR', 'Darwin')]
```

Optional extension

[65]:
```
college = {
    'Clare':{'abv':'CL','year':1326,'student':808},
    'Corpus Christi':{'abv':'CC','year':1352,'student':553},
    'Churchill':{'abv':'CHU','year':1960,'student':845},
    'Darwin':{'abv':'DAR','year':1964,'student':755},
    'Clare Hall':{'abv':'CLH','year':1966,'student':249}}
print(college)
#the greatest number of students and print the abbreviation;
max_students = max([int(i['student']) for i in college.values()])
print(max_students)

#the oldest
oldest = max([int(i['year']) for i in college.values()])
print(oldest)
```

```
{'Clare': {'abv': 'CL', 'year': 1326, 'student': 808}, 'Corpus Christi': {'abv':
'CC', 'year': 1352, 'student': 553}, 'Churchill': {'abv': 'CHU', 'year': 1960,
'student': 845}, 'Darwin': {'abv': 'DAR', 'year': 1964, 'student': 755}, 'Clare
Hall': {'abv': 'CLH', 'year': 1966, 'student': 249}}
845
1966
```