Ben Deatsman

Question 1: BMI

In [2]:
```
w = 150
h = 68

bmi = (w*703)/h**2
bmi
```

Out[2]:
```
22.80493079584775
```

Question 2: Math Library

In [3]:
```
import math
```

In [4]:
```
math.sqrt(14*.51)
```

Out[4]:
```
2.6720778431774774
```

In [5]:
```
math.pow(3.25, 2.784)
```

Out[5]:
```
26.61235308864194
```

Question 3: Cost of Pizza

In [10]:
```
pi = math.pi
d = 12
area = pi*(d/2)**2
area
```

Out[10]:
```
113.09733552923255
```

In [11]:
```
cost = 8
ppsi = cost/area
ppsi
```

Out[11]:
```
0.0707355302630646
```

In [13]:
```
d = 15
area = pi*(d/2)**2
area
```

Out[13]:
```
176.71458676442586
```

In [14]:
```
cost = 8
ppsi = cost/area
ppsi
```

Out[14]:
```
0.045270739368361346
```

Question 4: Conditional Statements and Snakes

In [16]:
```
language = 'python'

if language == 'python':
    print('I love snakes!')
elif language == 'R':
    print('Are you a pirate?')
else:
    print('What is language?')
```

```
I love snakes!
```

Question 5: Analyzing a vecotr of weights

In [17]:
```
weights = [9, 62, 57, 59, 59, 64, 56, 66, 67, 66]
```

In [19]:
```
weights * 2.20462
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[19], line 1
----> 1 weights * 2.20462

TypeError: can't multiply sequence by non-int of type 'float'
```

In [20]:
```
import numpy as np
```

In [24]:
```
np.array(weights)
```

Out[24]:
```
array([ 9, 62, 57, 59, 59, 64, 56, 66, 67, 66])
```

In [27]:
```
np.array(weights) * 2.20462
```

Out[27]:
```
array([ 19.84158, 136.68644, 125.66334, 130.07258, 130.07258, 141.09568,
        123.45872, 145.50492, 147.70954, 145.50492])
```

In [31]:
```
np.mean(weights) * 2.20462
```

Out[31]:
```
124.56102999999999
```

In [33]:
```
np.std(weights) * 2.20462
```

Out[33]:
```
35.864905604162125
```

Question 6: Back to BMI

In [34]:
```
heights = [62, 58, 61, 61, 59, 64, 63, 61, 60, 62]
np.array(heights)
```

Out[34]:
```
array([62, 58, 61, 61, 59, 64, 63, 61, 60, 62])
```

In [35]:
```
bmi = (np.array(weights)*703)/np.array(weights)**2
bmi
```

Out[35]:
```
array([78.11111111, 11.33870968, 12.33333333, 11.91525424, 11.91525424,
       10.984375  , 12.55357143, 10.65151515, 10.49253731, 10.65151515])
```

In [36]:
```
np.mean(bmi)
```

Out[36]:
```
18.094717664147463
```

Question 7: Nested Dictionary

In [45]:
```
d = {"a_list": [1, 2, 3,],
    "a_dict": {"first": ["this", "is", "inception"],
               "second": [1, 2, 3, "BANA"]}}
```

In [46]:
```
d["a_dict"]["second"][-1]
```

Out[46]:
```
'BANA'
```

Question 8: Sorting Arrays

In [48]:
```
from numpy.random import seed
from numpy.random import randint
seed(123)
weights = randint(low=50, high=100, size=20)
weights
```

Out[48]:
```
array([95, 52, 78, 84, 88, 67, 69, 92, 72, 83, 82, 99, 97, 59, 82, 96, 82,
       97, 75, 69])
```

In [50]:
```
weights.sort()
```

In [51]:
```
weights
```

Out[51]:
```
array([52, 59, 67, 69, 69, 72, 75, 78, 82, 82, 82, 83, 84, 88, 92, 95, 96,
       97, 97, 99])
```

In [56]:
```
weights[0:3]
```

Out[56]:
```
array([52, 59, 67])
```

Question 9: The Zen of Python

In [57]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

In [ ]:
```
```