Question 1:

In [1]:
```
x = [1, 2, 3]
y = 'bootcamp'
z = x + y
```

```
---------------------------------------------------------------------------
----
TypeError                                 Traceback (most recent call l
ast)
<ipython-input-1-5fb081dc0d7e> in <module>()
      1 x = [1, 2, 3]
      2 y = 'bootcamp'
----> 3 z = x + y

TypeError: can only concatenate list (not "str") to list
```

There is an arror because you cannot add two variables of different types

Question 2:

In [2]:
```
whos
```

```
Variable   Type    Data/Info
---------------------------
x          list    n=3
y          str     bootcamp
```

In [4]:
```
type(x)
```

Out[4]: list

In [5]:
```
type(y)
```

Out[5]: str

In [6]:
```
len(x)
```

Out[6]: 3

In [7]:
```
len(y)
```

Out[7]: 8

Question 3:

```
In [9]: type(2)
```

```
Out[9]: int
```

```
In [10]: type('2')
```

```
Out[10]: str
```

```
In [11]: type(2.0)
```

```
Out[11]: float
```

```
In [12]: type("2.0")
```

```
Out[12]: str
```

```
In [13]: type(2>1)
```

```
Out[13]: bool
```

```
In [14]: type('Itamar'>'Chase')
```

```
Out[14]: bool
```

```
In [15]: type([1,2])
```

```
Out[15]: list
```

```
In [17]: type((1, 2))
```

```
Out[17]: tuple
```

```
In [18]: type({1: 'one', 2: 'two'})
```

```
Out[18]: dict
```

Question 4:

```
In [21]: 1>=0
```

```
Out[21]: True
```

```
In [22]: 1 >= 0
```

```
Out[22]: True
```

```
In [23]: 1 > 1
```

```
Out[23]: False
```

```
In [24]: 1==1
```

```
Out[24]: True
```

In [25]:
```
1 == 1.0
```

Out[25]: True

In [26]:
```
'Spencer' == "Spencer"
```

Out[26]: True

In [27]:
```
2**3 > 3**2
```

Out[27]: False

In [29]:
```
1 >= 0 or 1 <= 2
```

Out[29]: True

In [30]:
```
1 >= 0 and 1 <= 2
```

Out[30]: True

Question 5:

In [33]:
```
if 2>1
  print('Yes, 2 is still greater than 1')
```

```
  File "<ipython-input-33-e53b371bbc74>", line 1
    if 2>1
         ^
SyntaxError: invalid syntax
```

This does not print because there is no : after 2>1 (aka invalid syntax)

In [34]:
```
if 2>1:
  print('Yes, 2 is still greater than 1')
```

```
Yes, 2 is still greater than 1
```

Question 6:

In [1]:
```
if True:
    print('on the one hand')
else:
    print('but on the other hand')
```

```
on the one hand
```

The result is 'on the one hand'.

In [2]:
```
if False:
    print('on the one hand')
else:
    print('but on the other hand')
```

```
but on the other hand
```

In [3]:
```
if not False:
    print('on the one hand')
else:
    print('but on the other hand')
```

```
on the one hand
```

Question 7:

In [5]:
```
cond = True
if cond:
    x = 'Chase'
else:
    x = 'Dave'
print(x)
```

```
Chase
```

Question 8:

In [6]:
```
x = [1, 2, 3, 4]
y = ['x', 'y', 'z']
```

In [7]:
```
if len(x)>len(y):
    print('x has more')
else:
    print("y at least has many")
```

```
x has more
```

Question 9:

Slicing separates lists by each item in the list and strings by each character. It is a good way to cut up types of data so you can access certain parts indevidually.

Question 10:

In [8]:
```
x = [1, 2, 3, 4, 5]
```

In [10]:
```
x[0]
```

Out[10]: 1

In [11]:
```
x[-1]
```

Out[11]: 5

In [21]:
```
x[0:4]
```

Out[21]: [1, 2, 3, 4]

Question 11:

In [22]:
```
sentance = 'This is a sentance: please slice it.'
```

In [43]:
```
sentance[0:4],sentance[5:7],sentance[8:9],sentance[10:18],sentance[20:26
],sentance[27:32],sentance[33:35],
```

Out[43]: ('This', 'is', 'a', 'sentance', 'please', 'slice', 'it')

Question 12:

In [44]:
```
x = [1, 2, "a", 'b', "fast", 'slow', 3, "Raghu", 'Liuren', 10]
```

a.

In [45]:
```
x[0]
```

Out[45]: 1

b.

In [47]:
```
x[3:7]
```

Out[47]: ['b', 'fast', 'slow', 3]

Question 13:

In [48]:
```
for item in x:
    print(item)
```

```
1
2
a
b
fast
slow
3
Raghu
Liuren
10
```

Question 14:

In [49]:
```
for item in x:
    if type(item)==str:
        print(item)
```

```
a
b
fast
slow
Raghu
Liuren
```

Question 15:

In [1]:
```
help(range)
```

```
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
 |
 |  Return an object that produces a sequence of integers from start (i
nclusive)
 |  to stop (exclusive) by step.  range(i, j) produces i, i+1, i+2,
..., j-1.
 |  start defaults to 0, and stop is omitted!  range(4) produces 0, 1,
2, 3.
 |  These are exactly the valid indices for a list of 4 elements.
 |  When step is given, it specifies the increment (or decrement).
 |
 |  Methods defined here:
 |
 |  __bool__(self, /)
 |      self != 0
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __new__(*args, **kwargs) from builtins.type
```

```
 |      Create and return a new object.  See help(type) for accurate si
gnature.
 |
 |  __reduce__(...)
 |      helper for pickle
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __reversed__(...)
 |      Return a reverse iterator.
 |
 |  count(...)
 |      rangeobject.count(value) -> integer -- return number of occurre
nces of value
 |
 |  index(...)
 |      rangeobject.index(value, [start, [stop]]) -> integer -- return
index of value.
 |      Raise ValueError if the value is not present.
 |
 |  ----------------------------------------------------------------------
---
 |  Data descriptors defined here:
 |
 |  start
 |
 |  step
 |
 |  stop
```

The range function...

In [2]:
```
range(3, 12, 2)
```

Out[2]:
```
range(3, 12, 2)
```

In [3]:
```
list(range(3, 12, 2))
```

Out[3]:
```
[3, 5, 7, 9, 11]
```

The range function range(3, 12, 2) created a list of numbers from 3 to 12 increasing by incriments of 2.

Question 16:

In [8]:
```
x = range(0, 30, 3)
a = sum(x)
print(a)
```

```
135
```

Question 17:

In [12]:
```
def pocket_change(x, y, z, n):
    if x or y or z or n == int:
        dollar = float(x/100)+float(y/20)+float(z/10)+float(n/4)
        print('$'+str(float(dollar)))
```

In [13]:
```
pocket_change(1, 2, 3, 4)
```

```
$1.41
```

Question 18:

In [45]:
```
def notsix(list):
    for item in list:
        if str(item)[0] == '6':
            print(item)
```

In [46]:
```
notsix([1234, 6783, 6, 4321, 9876])
```

```
6783
6
```

Question 19:

In [51]:
```
old_list = [1234, 6783, 6, 4321, 9876]
new_list = [x for x in old_list if str(x)[0] !="6"]
```

In [52]:
```
old_list
```

Out[52]:
```
[1234, 6783, 6, 4321, 9876]
```

In [53]:
```
new_list
```

Out[53]:
```
[1234, 4321, 9876]
```

This is eliminating any item in old_list that begins with the number 6

Question 20:

In [54]:
```
z = {1: 'one', 2: 'two', 3: 'three'}
```

a.

```
In [55]: type(z)
```

```
Out[55]: dict
```

z is a dict (dictionary. This means it is esspecilly a mini- database defining certain objects, in this case defining integers by their written string counter parts.

```
In [56]: len(z)
```

```
Out[56]: 3
```

b.

The integers to the left of the : are keys and the strings to the right of the : are values.

c.

```
In [60]: z[2]
```

```
Out[60]: 'two'
```

d.

In [62]:
```
help(z.keys())
```

```
Help on dict_keys object:

class dict_keys(object)
 |  Methods defined here:
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __or__(self, value, /)
 |      Return self|value.
 |
 |  __rand__(self, value, /)
 |      Return value&self.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __ror__(self, value, /)
 |      Return value|self.
 |
 |  __rsub__(self, value, /)
 |      Return value-self.
 |
 |  __rxor__(self, value, /)
 |      Return value^self.
 |
 |  __sub__(self, value, /)
```

```
 |      Return self-value.
 |
 |  __xor__(self, value, /)
 |      Return self^value.
 |
 |  isdisjoint(...)
 |      Return True if the view and the given iterable have a null inte
rsection.
 |
 |  ----------------------------------------------------------------------
---
 |  Data and other attributes defined here:
 |
 |  __hash__ = None
```

In [63]:
```
help(z.values())
```

```
Help on dict_values object:

class dict_values(object)
 |  Methods defined here:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __repr__(self, /)
 |      Return repr(self).
```

In [65]:
```
z.keys()
```

Out[65]:
```
dict_keys([1, 2, 3])
```

In [66]:
```
z.values()
```

Out[66]:
```
dict_values(['one', 'two', 'three'])
```

e.

In [67]:
```
list(z.keys())
```

Out[67]:
```
[1, 2, 3]
```

f.

```
In [68]: list(z.values())
```

```
Out[68]: ['one', 'two', 'three']
```

g.

```
In [69]: list(z)
```

```
Out[69]: [1, 2, 3]
```

Question 21:

Aprox. 90 minutes.