ch.11

October 12, 2020

[10]:
```
# Ch. 11: Lists
```

[11]:
```
# Ch. 11 Exercises
```

[12]:
```
# Ex. 1
```

[13]:
```
print(list(range(10, 0, -2)))
```

```
[10, 8, 6, 4, 2]
```

[14]:
```
# Ex. 2
```

[15]:
```
from unit_tester import test
import turtle

tess = turtle.Turtle()
alex = tess
alex.color("hotpink")

#this is aliacing. It creates one turtle instance. Yes setting the color of alex
#also changes color of tess
```

[16]:
```
# Ex. 3

a = [1, 2, 3]
b = a[:]
print(a is b)
print(a == b)
print(b)
b[0] = 5
print(a is b)
print(a == b)
print(b)
```

```
False
True
[1, 2, 3]
False
```

```
False
[5, 2, 3]
```

[17]:
```
#Ex. 4



this = ["I", "am", "not", "a", "crook"]
that = ["I", "am", "not", "a", "crook"]

print("Test 1: {0}".format(this is that))
that = this   #here "that" and "this" both reffer to the same object
print("Test 2: {0}".format(this is that))
```

```
Test 1: False
Test 2: True
```

[18]:
```
#Ex. 5

def add_vectors(u, v):
    list = []
    for i in range(len(u)):
        new_elem = u[i] + v[i]
        list.append(new_elem)
        print(list)
    return list

test(add_vectors([1, 1], [1, 1]) == [2, 2])
test(add_vectors([1, 2], [1, 4]) == [2, 6])
test(add_vectors([1, 2, 1], [1, 4, 3]) == [2, 6, 4])
```

```
[2]
[2, 2]
Test at line 11 ok.
[2]
[2, 6]
Test at line 12 ok.
[2]
[2, 6]
[2, 6, 4]
Test at line 13 ok.
```

[19]:
```
#Ex. 6

def scalar_mult(s, v):
    list = []
    for i in range(len(v)):
        new_elem = s* v[i]
        list.append(new_elem)
        print(list)
    return list


test(scalar_mult(5, [1, 2]) == [5, 10])
test(scalar_mult(3, [1, 0, -1]) == [3, 0, -3])
test(scalar_mult(7, [3, 0, 5, 11, 2]) == [21, 0, 35, 77, 14])
```

```
[5]
[5, 10]
Test at line 12 ok.
[3]
[3, 0]
[3, 0, -3]
Test at line 13 ok.
[21]
[21, 0]
[21, 0, 35]
[21, 0, 35, 77]
[21, 0, 35, 77, 14]
Test at line 14 ok.
```

[20]:
```
#Ex. 7

def dot_product(u, v):
    list = []
    b = 0
    for i in range(len(v)):
        new_elem = u[i] * v[i]
        list.append(new_elem)
        print(list)
    for i in range(len(list)):
        b = b + list[i]
        print(b)
    return b


test(dot_product([1, 1], [1, 1]) == 2)
test(dot_product([1, 2], [1, 4]) == 9)
test(dot_product([1, 2, 1], [1, 4, 3]) == 12)
```

```
[1]
[1, 1]
1
2
Test at line 16 ok.
[1]
[1, 8]
```

```
1
9
Test at line 17 ok.
[1]
[1, 8]
[1, 8, 3]
1
9
12
Test at line 18 ok.
```

[21]:
```python
#Ex. 8

#skipped
```

[22]:
```python
#Ex. 9

song = "The rain in Spain..."

print(song.split())
print(" ".join(song.split())) # makes a split first, then uses the empty spaces␣
 ↪to join
print(song) # is the same as above
```

```
['The', 'rain', 'in', 'Spain...']
The rain in Spain...
The rain in Spain...
```

[23]:
```python
#Ex. 10

def replace(s, old, new):
    new_elem = new.join(s.split(old))
    return new_elem

test(replace("Mississippi", "i", "I") == "MIssIssIppI")

s = "I love spom! Spom is my favorite food. Spom, spom, yum!"

test(replace(s, "om", "am") == "I love spam! Spam is my favorite food. Spam,␣
 ↪spam, yum!")

test(replace(s, "o", "a") == "I lave spam! Spam is my favarite faad. Spam, spam,␣
 ↪yum!")
```

```
Test at line 7 ok.
Test at line 11 ok.
Test at line 13 ok.
```

[24]:
```
#Ex. 11

def swap(x, y): # Incorrect version
    print("before swap statement: x:", x, "y:", y)
    (x, y) = (y, x)
    print("after swap statement: x:", x, "y:", y)

a = ["This", "is", "fun"]
b = [2,3,4]
print("before swap function call: a:", a, "b:", b)
swap(a, b)
print("after swap function call: a:", a, "b:", b)

#the values of a and b didn't change
# a modifier is neccessery here to change the values of a and b
```

```
before swap function call: a: ['This', 'is', 'fun'] b: [2, 3, 4]
before swap statement: x: ['This', 'is', 'fun'] y: [2, 3, 4]
after swap statement: x: [2, 3, 4] y: ['This', 'is', 'fun']
after swap function call: a: ['This', 'is', 'fun'] b: [2, 3, 4]
```