ENPM 661 - Planning for Autonomous Robots

Project 0 - Introduction to Python

Diane Ngo 2/1/21

1.0 Lists

A list is a sequence of values. Lists need not be homogeneous always which makes it a most powerful tool in Python. Lists are mutable, and hence, they can be altered even after their creation. Lists in Python are ordered and have a definite count.The values in the list, also known as elements are enclosed in square brackets.For example : [25, 40, 50, 80]

1.1 Creating a List

In [73]:
```
#Creating a list of numbers and printing the second element:
num_list = [0, 1, 2, 3, 4, 5]
num_list_2 = [1, 3, 5, 7, 9]

# Accessing and printing the second element
print(num_list[1])
print(num_list_2[2])
# Indexing starts with 0!
```

```
1
5
```

1.2 Modifying the List

In [74]:
```
#Modifying the first element and printing the updating list
num_list[0] = 100
num_list_2[0] = 13
print(num_list)
print(num_list_2)
```

```
[100, 1, 2, 3, 4, 5]
[13, 3, 5, 7, 9]
```

1.3 Adding a new element to the List

In [75]:
```
# Adding number 10
num_list.append(10)
print(num_list)
num_list_2.append(21)
print(num_list_2)
```

```
[100, 1, 2, 3, 4, 5, 10]
[13, 3, 5, 7, 9, 21]
```

1.4 Removing an element from the List

In [76]:
```
#Removing the last element
# Method 1 using the in-built pop function and accessing the last ele
ment with negative index:
num_list.pop(-1)
print(num_list)
num_list_2.pop(-1)
print(num_list_2)
```

```
[100, 1, 2, 3, 4, 5]
[13, 3, 5, 7, 9]
```

In [77]:
```
# Method 2 by finding the index of the last element and deleting it:
del num_list[len(num_list)-1]
print(num_list)
```

```
[100, 1, 2, 3, 4]
```

1.5 Sorting the list

In [78]:
```
num_list.sort()
print(num_list)
```

```
[1, 2, 3, 4, 100]
```

1.6 Slicing the list

Lets print the second and third element

In [79]:
```
print(num_list[1:3])
```

```
[2, 3]
```

Practice :

Create an empty list.\ Append the list using a for-loop with squares of numbers from 1 to 10.\ Sort and reverse the list.\ Delete the 7th and 8th element and print the list.

In [80]:
```
import math
my_list = []
for i in range(10):
    i = i+1
    i = math.pow(i, 2)
    my_list.append(i)
print(my_list)
my_list.sort(reverse=True)
print(my_list)
my_list.pop(-3)
my_list.pop(-3)
print(my_list)
```

```
[1.0, 4.0, 9.0, 16.0, 25.0, 36.0, 49.0, 64.0, 81.0, 100.0]
[100.0, 81.0, 64.0, 49.0, 36.0, 25.0, 16.0, 9.0, 4.0, 1.0]
[100.0, 81.0, 64.0, 49.0, 36.0, 25.0, 4.0, 1.0]
```

2.0 Tuples

A Tuple is a collection of Python objects separated by commas. In someways a tuple is similar to a list in terms of indexing, nested objects and repetition but a tuple is immutable unlike lists which are mutable.

2.1 Creating tuples

In [81]:
```
# An empty tuple
empty_tuple = ()
print (empty_tuple)
```

```
()
```

In [82]:
```
# Creating non-empty tuples

# One way of creation
tup = 'python', 'cplus'
print(tup)

# Another for doing the same
tup = ('python', 'cplus')
print(tup)
```

```
('python', 'cplus')
('python', 'cplus')
```

2.2 Concatenation of Tuples

In [83]:
```
# Code for concatenating 2 tuples

tuple1 = (0, 1, 2, 3)
tuple2 = ('python', 'robot')

# Concatenating above two
print(tuple1 + tuple2)
```

```
(0, 1, 2, 3, 'python', 'robot')
```

2.3 Nesting of Tuples

In [84]:
```
# Code for creating nested tuples

tuple1 = (0, 1, 2, 3)
tuple2 = ('python', 'robot')
tuple3 = (tuple1, tuple2)
print(tuple3)
```

```
((0, 1, 2, 3), ('python', 'robot'))
```

2.4 Repetition in Tuples

In [85]:
```
# Code to create a tuple with repetition

tuple3 = ('python',)*3
print(tuple3)
```

```
('python', 'python', 'python')
```

2.5 Immutable Tuples

In [86]:
```
# Code to test that tuples are immutable (You will be getting an erro
r message)

tuple1 = (0, 1, 2, 3)
tuple1[0] = 4
print(tuple1)
```

```
---------------------------------------------------------------------
------
TypeError                                 Traceback (most recent call
last)
<ipython-input-86-0329ed6d72fe> in <module>
      2
      3 tuple1 = (0, 1, 2, 3)
----> 4 tuple1[0] = 4
      5 print(tuple1)

TypeError: 'tuple' object does not support item assignment
```

2.6 Slicing in Tuples

In [87]:
```
# Code to test slicing

tuple1 = (0 ,1, 2, 3)
print(tuple1[1:])
print(tuple1[::-1])
print(tuple1[2:4])
```

```
(1, 2, 3)
(3, 2, 1, 0)
(2, 3)
```

2.7 Deleting a Tuple

In [ ]:
```
# Code for deleting a tuple

tuple3 = (0, 1)
del tuple3
print(tuple3)
```

2.8 Finding Length of a Tuple

In [11]:
```
# Code for printing the length of a tuple

tuple2 = ('python', 'robot')
print(len(tuple2))
```

```
2
```

2.9 Converting list to a Tuple

In [13]:
```
# Code for converting a list and a string into a tuple

list1 = [0, 1, 2]
print(tuple(list1))
print(tuple('robot')) # string 'python'
```

```
(0, 1, 2)
('r', 'o', 'b', 'o', 't')
```

2.10 Tuples in a loop

In [ ]:
```
#python code for creating tuples in a loop

tup = ('ENPM661',)
n = 5  #Number of time loop runs
for i in range(int(n)):
    tup = (tup,)
    print(tup)
```

2.11 Using max() , min()

In [14]:
```
# A python program to demonstrate the use of
# cmp(), max(), min()

tuple1 = ('ENPM', '661')
tuple2 = ('planning', "robots")

print ('Maximum element in tuples 1,2: ' +
        str(max(tuple1)) +  ',' +
        str(max(tuple2)))
print ('Minimum element in tuples 1,2: ' +
     str(min(tuple1)) + ','  + str(min(tuple2)))
```

```
Maximum element in tuples 1,2: ENPM,robots
Minimum element in tuples 1,2: 661,planning
```

3.0 Dictionary

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a ‘comma’.

A Dictionary in Python works similar to the Dictionary in a real world. Keys of a Dictionary must be unique and of immutable data type such as Strings, Integers and tuples, but the key-values can be repeated and be of any type.

3.1 Creating a Dictionary

In Python, a Dictionary can be created by placing sequence of elements within curly {} braces, separated by ‘comma’. Dictionary holds a pair of values, one being the Key and the other corresponding pair element being its Key:value. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can’t be repeated and must be immutable.

Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing to

In [16]:
```
# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)

# Creating a Dictionary
# with Integer Keys
Dict = {1: 'Path', 2: 'Planning', 3: 'Robots'}
print("\nDictionary with the use of Integer Keys: ")
print(Dict)

# Creating a Dictionary
# with Mixed keys
Dict = {'Robot': 'Planning', 1: [1, 2, 3, 4]}
print("\nDictionary with the use of Mixed Keys: ")
print(Dict)

# Creating a Dictionary
# with dict() method
Dict = dict({1: 'Planning', 2: 'For', 3:'Robots'})
print("\nDictionary with the use of dict(): ")
print(Dict)

# Creating a Dictionary
# with each item as a Pair
Dict = dict([(1, 'Geeks'), (2, 'For')])
print("\nDictionary with each item as a pair: ")
print(Dict)
```

```
Empty Dictionary:
{}

Dictionary with the use of Integer Keys:
{1: 'Path', 2: 'Planning', 3: 'Robots'}

Dictionary with the use of Mixed Keys:
{'Robot': 'Planning', 1: [1, 2, 3, 4]}

Dictionary with the use of dict():
{1: 'Planning', 2: 'For', 3: 'Robots'}

Dictionary with each item as a pair:
{1: 'Geeks', 2: 'For'}
```

3.2 Nested Dictionary

In [17]:
```
# Creating a Nested Dictionary
# as shown in the below image
Dict = {1: 'Geeks', 2: 'For',
        3:{'A' : 'Welcome', 'B' : 'To', 'C' : 'Geeks'}}

print(Dict)
```

```
{1: 'Geeks', 2: 'For', 3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}
```

3.3 Adding elements to a Dictionary

In Python Dictionary, addition of elements can be done in multiple ways. One value at a time can be added to a Dictionary by defining value along with the key e.g. Dict[Key] = 'Value'. Updating an existing value in a Dictionary can be done by using the built-in update() method. Nested key values can also be added to an existing Dictionary.

Note- While adding a value, if the key value already exists, the value gets updated otherwise a new Key with the value is added to the Dictionary.

In [18]:
```
# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)

# Adding elements one at a time
Dict[0] = 'Robot'
Dict[2] = 'Planning'
Dict[3] = 1
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Adding set of values
# to a single Key
Dict['Value_set'] = 2, 3, 4
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Updating existing Key's Value
Dict[2] = 'Welcome'
print("\nUpdated key value: ")
print(Dict)

# Adding Nested Key value to Dictionary
Dict[5] = {'Nested' :{'1' : 'Life', '2' : 'Geeks'}}
print("\nAdding a Nested Key: ")
print(Dict)
```

```
Empty Dictionary:
{}

Dictionary after adding 3 elements:
{0: 'Robot', 2: 'Planning', 3: 1}

Dictionary after adding 3 elements:
{0: 'Robot', 2: 'Planning', 3: 1, 'Value_set': (2, 3, 4)}

Updated key value:
{0: 'Robot', 2: 'Welcome', 3: 1, 'Value_set': (2, 3, 4)}

Adding a Nested Key:
{0: 'Robot', 2: 'Welcome', 3: 1, 'Value_set': (2, 3, 4), 5: {'Neste
d': {'1': 'Life', '2': 'Geeks'}}}
```

3.4 Accessing elements from a Dictionary

In order to access the items of a dictionary refer to its key name.Key can be used inside square brackets.There is also a method called get() that will also help in acessing the element from a dictionary.

In [19]:
```
# Python program to demonstrate
# accesing a element from a Dictionary

# Creating a Dictionary
Dict = {1: 'Planning', 'name': 'For', 3: 'Robots'}

# accessing a element using key
print("Acessing a element using key:")
print(Dict['name'])

# accessing a element using key
print("Acessing a element using key:")
print(Dict[1])

# accessing a element using get()
# method
print("Acessing a element using get:")
print(Dict.get(3))
```

```
Acessing a element using key:
For
Acessing a element using key:
Planning
Acessing a element using get:
Robots
```

3.5 Removing Elements from Dictionary

In Python Dictionary, deletion of keys can be done by using the del keyword. Using del keyword, specific values from a dictionary as well as whole dictionary can be deleted. Other functions like pop() and popitem() can also be used for deleting specific values and arbitrary values from a Dictionary. All the items from a dictionary can be deleted at once by using clear() method. Items in a Nested dictionary can also be deleted by using del keyword and providing specific nested key and particular key to be deleted from that nested Dictionary. Note- del Dict will delete the entire dictionary and hence printing it after deletion will raise an Error.

In [ ]:
```
# Initial Dictionary
Dict = { 5 : 'Welcome', 6 : 'To', 7 : 'Geeks',
        'A' : {1 : 'Geeks', 2 : 'For', 3 : 'Geeks'},
        'B' : {1 : 'Geeks', 2 : 'Life'}}
print("Initial Dictionary: ")
print(Dict)

# Deleting a Key value
del Dict[6]
print("\nDeleting a specific key: ")
print(Dict)

# Deleting a Key from
# Nested Dictionary
del Dict['A'][2]
print("\nDeleting a key from Nested Dictionary: ")
print(Dict)

# Deleting a Key
# using pop()
Dict.pop(5)
print("\nPopping specific element: ")
print(Dict)

# Deleting an arbitrary Key-value pair
# using popitem()
Dict.popitem()
print("\nPops an arbitrary key-value pair: ")
print(Dict)

# Deleting entire Dictionary
Dict.clear()
print("\nDeleting Entire Dictionary: ")
print(Dict)
```

3.6 Dictionary Methods

3.6.1 Copy()

They copy() method returns a shallow copy of the dictionary.

Syntax

dict.copy()

In [21]:
```
original = {1:'robots', 2:'planning'}
print(original)
## Write code to copy "original" dictionary and print the newly copie
d dictionary
new_copy = original.copy()
print(new_copy)
```

```
{1: 'robots', 2: 'planning'}
{1: 'robots', 2: 'planning'}
```

3.6.2 clear()

The clear() method removes all items from the dictionary.

Syntax

dict.clear()

In [22]:
```
text = {1: "geeks", 2: "for"}

# write the code to clear the content of the dictionary and then prin
t the empty dictionary
text.clear()
print(text)
```

```
{}
```

3.6.3 pop()

Python pop() method removes an element from the dictionary. It removes the element which is associated to the specified key.

If specified key is present in the dictionary, it remove and return its value.

If the specified key is not present, it throws an error KeyError.

Syntax

dict.pop(key, def)

Parameters :

key : The key whose key-value pair has to be returned and removed.

def : The default value to return if specified key is not present.

In [25]:
```
# Python dictionary pop() Method
# Creating a dictionary
inventory = {'shirts': 25, 'paints': 220, 'shoes': 525, 'tshirts': 21
7}
my_inventory = {'gpu' : 1, 'cpu': 2, 'psu':3, 'mouse':4, 'keyboard':5
, 'monitor':6}
# Calling method
element = inventory.pop('shoes')
my_element = my_inventory.pop('cpu')
# Displaying result
print(element)
print(my_element)
```

```
525
2
```

In [26]:
```
inventory = {'shirts': 25, 'paints': 220, 'tshirts': 217}

# write code to get the value for the key 'shoes'. If the key is not
 in the dictionary return 100
inventory.pop('shoes', 100)
```

Out[26]: 100

3.6.4 popitem()

popitem() method in dictionary helps to achieve similar purpose. It removes the arbitrary key-value pair from the dictionary and returns it as tuple.

Syntax

dict.popitem()

Parameters : None

Returns : A tuple containing the arbitrary key-value pair from dictionary. That pair is removed from dictionary.

In [28]:
```
test_dict = { "Nikhil" : 7, "Akshat" : 1, "Akash" : 2 }

# write the code using popitem() to return + remove arbitrary key val
ue pair and print it
pair = test_dict.popitem()
print(test_dict)
print(pair)
```

```
{'Nikhil': 7, 'Akshat': 1}
('Akash', 2)
```

3.6.5 get()

This method returns the value for the given key, if present in the dictionary. If not, then it will return None (if get() is used with only one argument).

Syntax

Dict.get(key, default=None)

In [36]:
```
dic = {"A":1, "B":2}
print(dic.get("A"))

# write the code to get the value for key 'C'. Print "Not found" if t
he key is not present
default="Not Found"
dic.get("C", default)
```

```
1
```

Out[36]:
```
'Not Found'
```

3.6.6 values()

values() is an inbuilt method in Python programming language that returns a list of all the values available in a given dictionary.

Syntax

dictionary_name.values()

In [ ]:
```
dictionary = {"A": 2, "B": 3, "C": 4}
# write the code to print the values in the dictionary
```

3.6.7 update()

In Python Dictionary, update() method updates the dictionary with the elements from the another dictionary object or from an iterable of key/value pairs.

Syntax

dict.update([other_dictionary])

Parameters: This method takes either a dictionary or an iterable object of key/value pairs (generally tuples) as parameters.

Returns: It doesn't return any value but updates the Dictionary with elements from a dictionary object or an iterable object of key/value pairs.

In [39]:
```
# Dictionary with three items
Dictionary1 = { 'A': 'Path', 'B': 'Planning', }
Dictionary2 = { 'B': 'Robots' }

# Dictionary before Updation
print("Original Dictionary:")
print(Dictionary1)

# Write the code to update "Dictionary1" with "Dictionary2" and print
Dictionary1.update(Dictionary2)
print(Dictionary1)
```

```
Original Dictionary:
{'A': 'Path', 'B': 'Planning'}
{'A': 'Path', 'B': 'Robots'}
```

4.0 Loop: 'for'

Creating a new List with the name "my_list"

In [41]:
```
my_list = [1, 2, 3, 4, 'Python', 'is', 'cool']
```

Iterating through each term in the List. Printing each element of the List

In [40]:
```
for item in my_list:
    print(item)
```

```
100.0
81.0
64.0
49.0
36.0
25.0
4.0
1.0
```

Syntax: "break"

Iterating through the list and if target value is reached, break.

This will break and terminate the loop when the required keyword is found.

In [42]:
```
for item in my_list:
    if item == 'Python':
        break
    print(item)
```

```
1
2
3
4
```

Syntax "continue"

Iterating through the list, if target is reached, move on to the next element

The number '4' doesn't get printed

Continue to the next item without executing the lines occuring after continue inside the loop.

In [43]:
```
for item in my_list:
    if item == 4:
        continue
    print(item)
```

```
1
2
3
Python
is
cool
```

Syntax "enumerate"

Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object.

This enumerate object can then be used directly in for loops

In [50]:
```
for idx, val in enumerate(my_list):
    print('idx: {}, value: {}'.format(idx, val))
```

```
idx: 0, value: 1
idx: 1, value: 2
idx: 2, value: 3
idx: 3, value: 4
idx: 4, value: Python
idx: 5, value: is
idx: 6, value: cool
```

Syntax "range"

To control the iterations of "for loop", range can be used.

In [47]:
```
for number in range(5):
    print(number)
```

```
0
1
2
3
4
```

In [48]:
```
for number in range(2, 5):
    print(number)
```

```
2
3
4
```

The last number '2' specifies the step size of each iteration

In [49]:
```
for number in range(0, 10, 2):
    print(number)
```

```
0
2
4
6
8
```

Create a list of strings based on a list of numbers

The rules:

If the number is a multiple of five and odd, the string should be 'five odd'

If the number is a multiple of five and even, the string should be 'five even'

If the number is odd, the string is 'odd'

If the number is even, the string is 'even'

(P.S. Refer textbook for mathematical operations - Section 2.5)

In [ ]:
```
numbers = [1, 3, 4, 6, 81, 80, 100, 95]
```

In [68]:
```
numbers = [1, 3, 4, 6, 81, 80, 100, 95]
this_list = []
# Your implementation
for item in numbers:
    if (item % 2) == 0:
        this_list.append("even")
        if (item % 5) == 0:
            this_list.pop()
            this_list.append("five even")
    else:
        this_list.append('odd')
        if (item % 5) == 0:
            this_list.pop()
            this_list.append("five odd")

print(this_list)
```

```
['odd', 'odd', 'even', 'even', 'odd', 'five even', 'five even', 'five
odd']
```

Assert will cross check if the output generated is the required ouput. If not, it will throw an error

In [70]:
```
assert this_list == ['odd', 'odd', 'even', 'even', 'odd', 'five even'
, 'five even', 'five odd']
```

Loop: 'while'

With the while loop we can execute a set of statements as long as a condition is true.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

In [ ]:
```
i = 1
while i < 6:
    print(i)
    i += 1
```

With the break statement we can stop the loop even if the while condition is true:

In [ ]:
```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

With the continue statement we can stop the current iteration, and continue with the next:

In [ ]:
```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

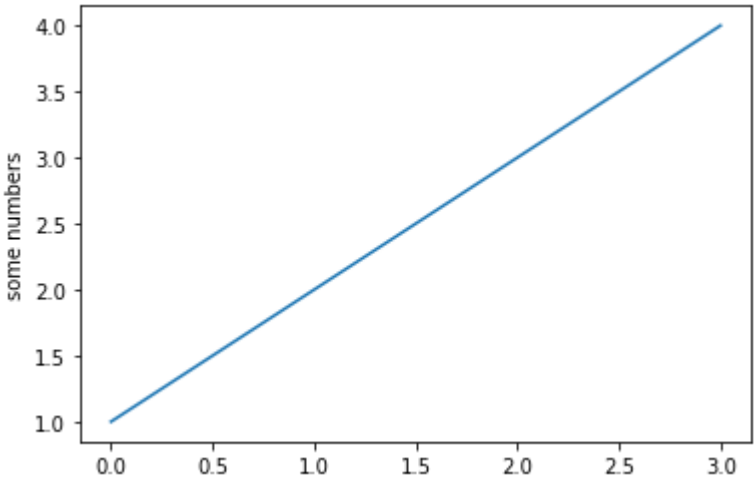
With the else statement we can run a block of code once when the condition no longer is true:

In [ ]:
```
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")
```

Plotting using Matplotlib

matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

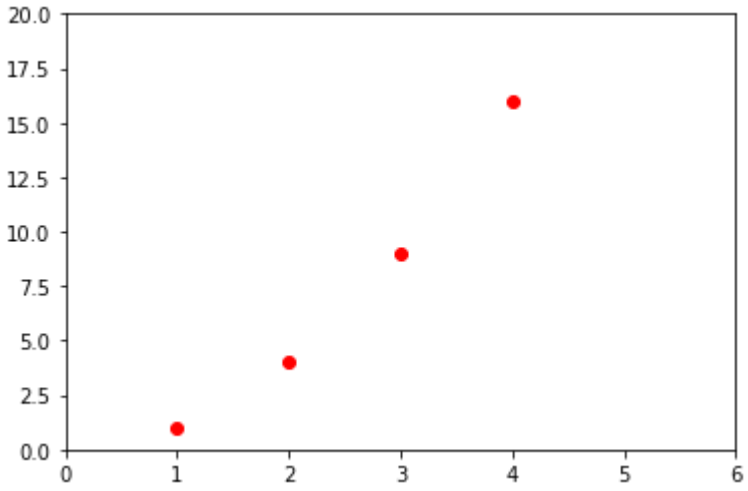
In [44]:
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('Scale:some numbers')
plt.show()
```



Formatting the style of your plot

For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot. The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string. The default format string is 'b-', which is a solid blue line. For example, to plot the above with red circles, you would issue

In [45]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

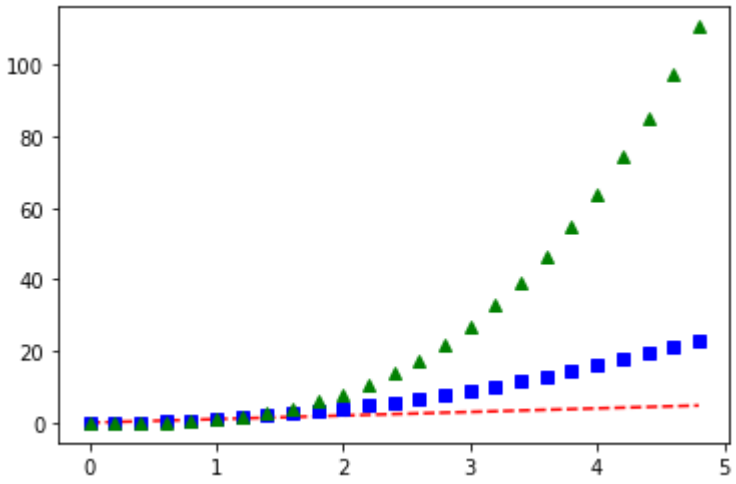


If matplotlib were limited to working with lists, it would be fairly useless for numeric processing. Generally, you will use numpy arrays. In fact, all sequences are converted to numpy arrays internally. The example below illustrates a plotting several lines with different format styles in one command using arrays.

In [46]:
```
import numpy as np

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



In [ ]:
```

```