Raspberry Pi for Beginners

LESSON 5

MAKERHOUSE

EMPOWERING MAKERS

TABLE OF CONTENTS

PYTHON DATA STRUCTURES

1 INTRODUCTION

Python allows us to store data – strings, numbers – together to create structures either in sequential or non-sequential as depicted in Figure 1.

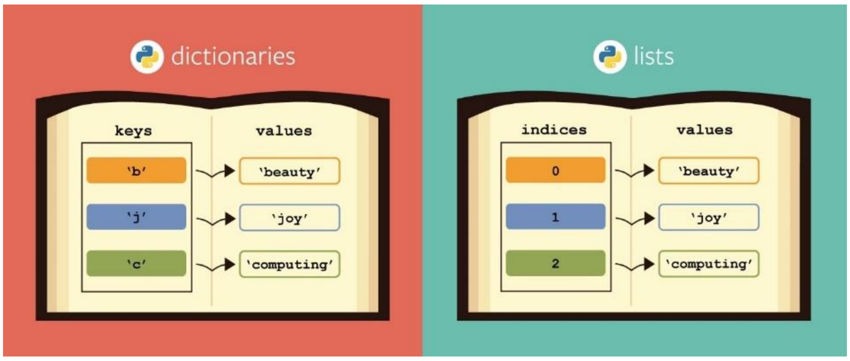
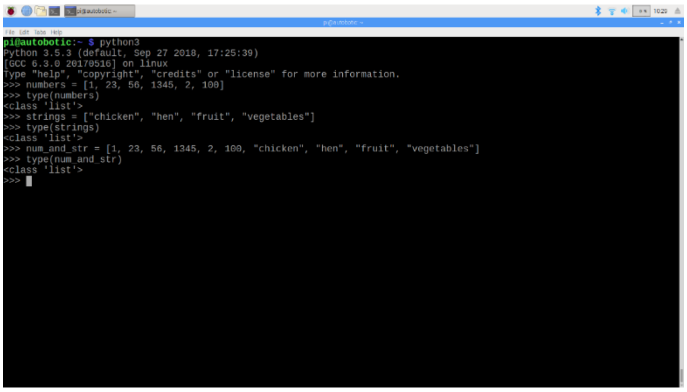


Figure 1: Comparing Dictionaries and Lists

1.1 LISTS AND TUPLES

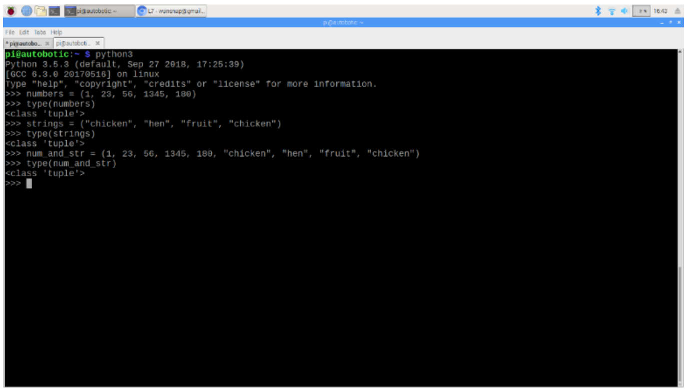
Both, in general, were almost identical; similar – store information one piece after the next – in orderly -- sequences. It's the simplest method to construct a structure in Python.

1.1.1 CREATE



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = [1, 23, 56, 1345, 2, 100]
>>> type(numbers)
<class 'list'>
>>> strings = ["chicken", "hen", "fruit", "vegetables"]
>>> type(strings)
<class 'list'>
>>> num_and_str = [1, 23, 56, 1345, 2, 100, "chicken", "hen", "fruit", "vegetables"]
>>> type(num_and_str)
<class 'list'>
>>> 
```

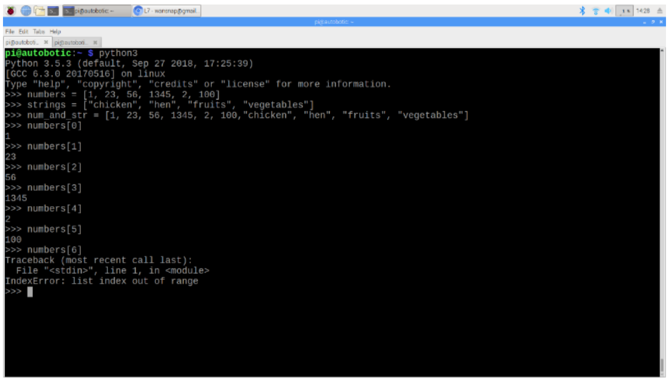
Figure 2a: Python Lists create. The main characteristics of Python Lists – [] – enclosing with a square bracket. A list can have a number (int/float) only, string (str) or even of both combinations.



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = (1, 23, 56, 1345, 180)
>>> type(numbers)
<class 'tuple'>
>>> strings = ("chicken", "hen", "fruit", "chicken")
>>> type(strings)
<class 'tuple'>
>>> num_and_str = (1, 23, 56, 1345, 180, "chicken", "hen", "fruit", "chicken")
>>> type(num_and_str)
<class 'tuple'>
>>> 
```

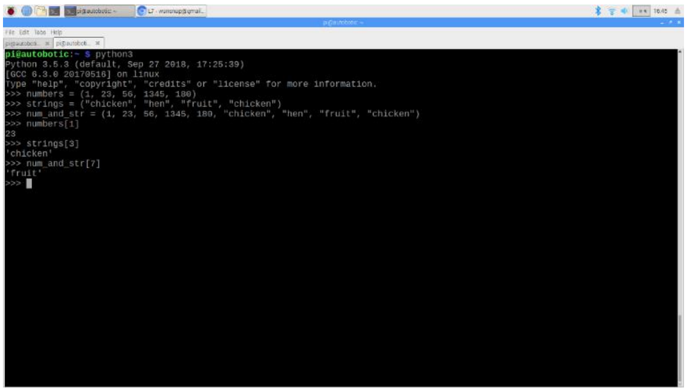
Figure 2b: Python tuple create. T the main characteristics of Python Tuples – () – enclosing with a bracket. Like list, tuples also can have a number (int/float) only, string (str) or even of both combinations.

1.1.2 ACCESSING



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = [1, 23, 56, 1345, 2, 100]
>>> strings = ["chicken", "hen", "fruits", "vegetables"]
>>> num_and_str = [1, 23, 56, 1345, 2, 100,"chicken", "hen", "fruits", "vegetables"]
>>> numbers[0]
1
>>> numbers[1]
23
>>> numbers[2]
56
>>> numbers[3]
1345
>>> numbers[4]
2
>>> numbers[5]
100
>>> numbers[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> 
```

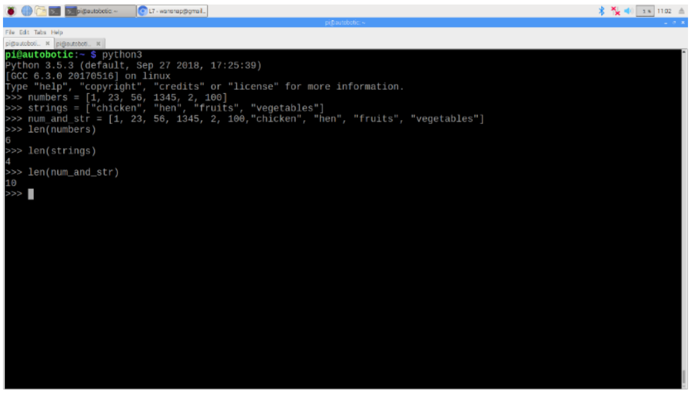
Figure 3a: Retrieving list items using – [] – square bracket with index.



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = (1, 23, 56, 1345, 180)
>>> strings = ("chicken", "hen", "fruit", "chicken")
>>> num_and_str = (1, 23, 56, 1345, 180, "chicken", "hen", "fruit", "chicken")
>>> numbers[1]
23
>>> strings[3]
'chicken'
>>> num_and_str[7]
'fruit'
>>> 
```

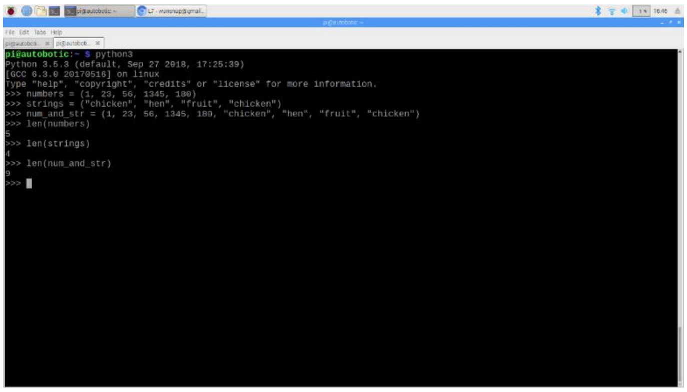
Figure 3b: Retrieving tuple items using – [] – square bracket with index.

1.1.3 LENGTH



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = [1, 23, 56, 1345, 2, 100]
>>> strings = ["chicken", "hen", "fruits", "vegetables"]
>>> num_and_str = [1, 23, 56, 1345, 2, 100,"chicken", "hen", "fruits", "vegetables"]
>>> len(numbers)
6
>>> len(strings)
4
>>> len(num_and_str)
10
>>>
```

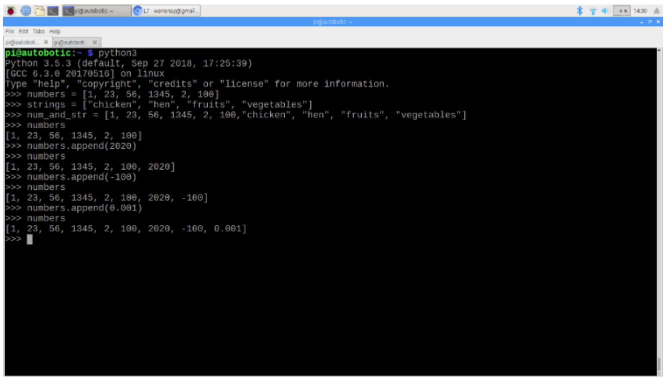
Figure 4a: Checking how many elements there are in a list using **len()** function



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = (1, 23, 56, 1345, 180)
>>> strings = ("chicken", "hen", "fruit", "chicken")
>>> num_and_str = (1, 23, 56, 1345, 180, "chicken", "hen", "fruit", "chicken")
>>> len(numbers)
5
>>> len(strings)
4
>>> len(num_and_str)
9
>>>
```

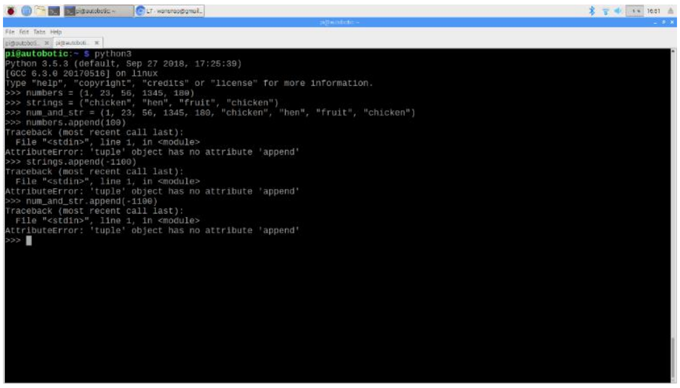
Figure 4b: Checking how many elements there are in a tuple using **len()** function

1.1.4 ADDING ELEMENT



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = [1, 23, 56, 1345, 2, 100]
>>> strings = ["chicken", "hen", "fruits", "vegetables"]
>>> num_and_str = [1, 23, 56, 1345, 2, 100,"chicken", "hen", "fruits", "vegetables"]
>>> numbers
[1, 23, 56, 1345, 2, 100]
>>> numbers.append(2020)
>>> numbers
[1, 23, 56, 1345, 2, 100, 2020]
>>> numbers.append(-100)
>>> numbers
[1, 23, 56, 1345, 2, 100, 2020, -100]
>>> numbers.append(0.001)
>>> numbers
[1, 23, 56, 1345, 2, 100, 2020, -100, 0.001]
>>>
```

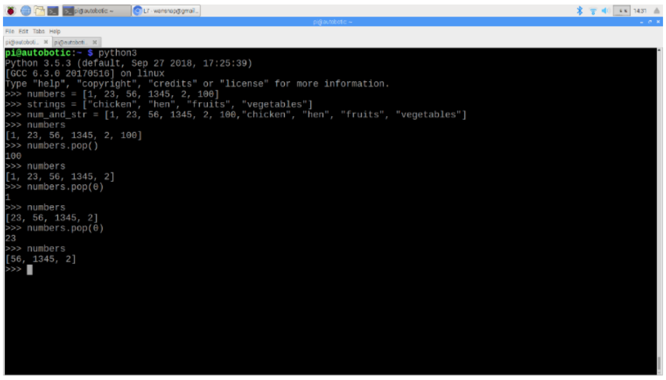
Figure 5a: Add new item in list using **append()** function



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = (1, 23, 56, 1345, 180)
>>> strings = ("chicken", "hen", "fruit", "chicken")
>>> num_and_str = (1, 23, 56, 1345, 180, "chicken", "hen", "fruit", "chicken")
>>> numbers.append(100)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>> strings.append(-1100)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>> num_and_str.append(-1100)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>> 
```

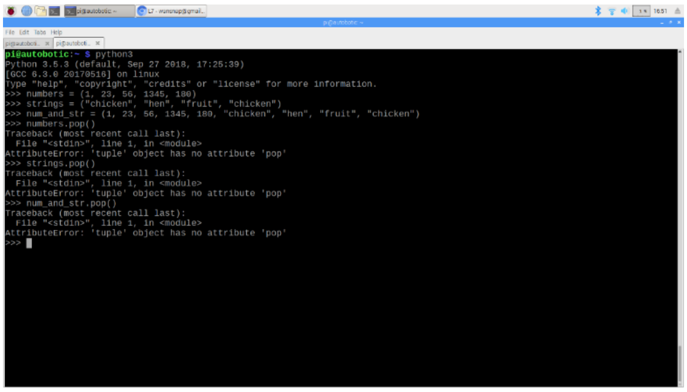
Figure 5b: Cannot add new item on tuple – reinitialize – require to create a new set of tuple

1.1.5 REMOVING ELEMENT



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = [1, 23, 56, 1345, 2, 100]
>>> strings = ["chicken", "hen", "fruits", "vegetables"]
>>> num_and_str = [1, 23, 56, 1345, 2, 100,"chicken", "hen", "fruits", "vegetables"]
>>> numbers
[1, 23, 56, 1345, 2, 100]
>>> numbers.pop()
100
>>> numbers
[1, 23, 56, 1345, 2]
>>> numbers.pop(0)
1
>>> numbers
[23, 56, 1345, 2]
>>> numbers.pop(0)
23
>>> numbers
[56, 1345, 2]
>>> 
```

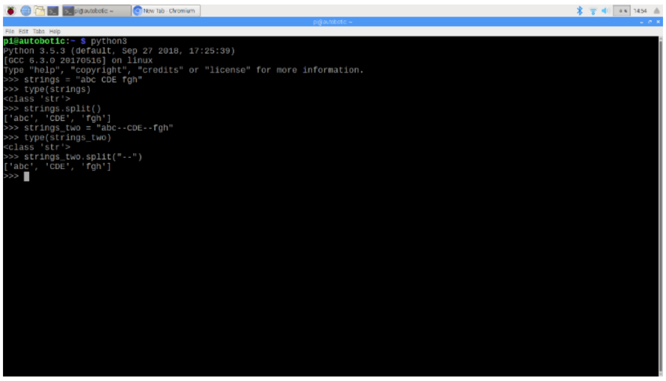
Figure 6a: Remove an item in list using – **pop()** – function



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = (1, 23, 56, 1345, 180)
>>> strings = ("chicken", "hen", "fruit", "chicken")
>>> num_and_str = (1, 23, 56, 1345, 180, "chicken", "hen", "fruit", "chicken")
>>> numbers.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'pop'
>>> strings.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'pop'
>>> num_and_str.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'pop'
>>> 
```

Figure 6b: Cannot remove an item on tuple – reinitialize – require to create a new set of tuple

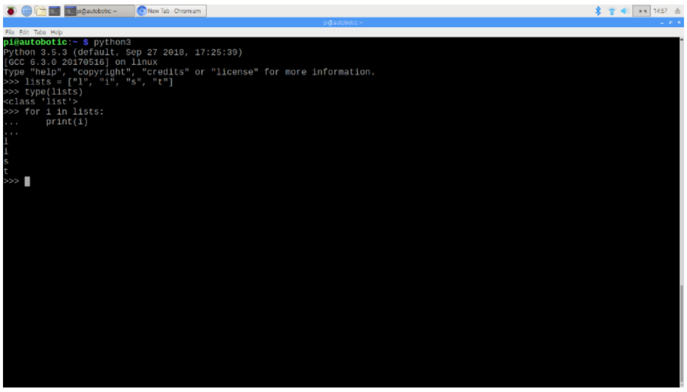
1.1.6 CREATE LIST BY PARSING A STRING



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> strings = "abc CDE fgh"
>>> type(strings)
<class 'str'>
>>> strings.split()
['abc', 'CDE', 'fgh']
>>> strings_two = "abc--CDE--fgh"
>>> type(strings_two)
<class 'str'>
>>> strings_two.split("--")
['abc', 'CDE', 'fgh']
>>> 
```

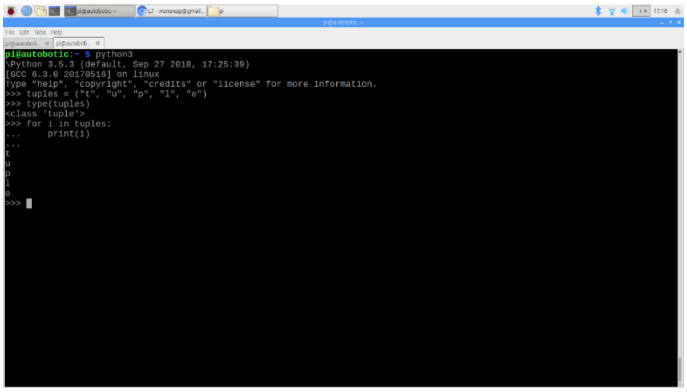
Figure 7: Create a new list by parsing a string – **split()** – function

1.1.7 ITERATE



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> lists = ["l", "i", "s", "t"]
>>> type(lists)
<class 'list'>
>>> for i in lists:
...     print(i)
...
l
i
s
t
>>> 
```

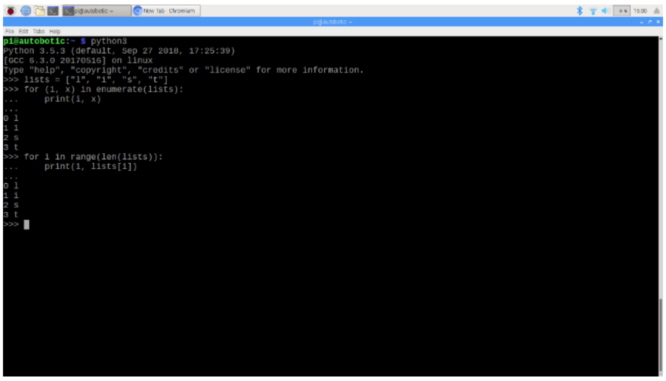
Figure 8a: Iterating over the list – **for** – loops



```
pi@autobotic:~ $ python3
\Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> tuples = ("t", "u", "p", "l", "e")
>>> type(tuples)
<class 'tuple'>
>>> for i in tuples:
...     print(i)
...
t
u
p
l
e
>>> 
```

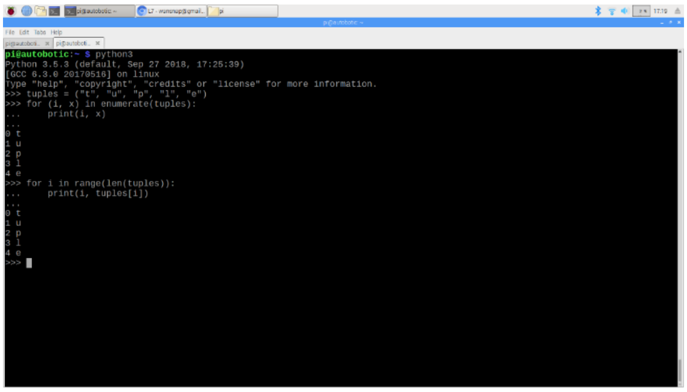
Figure 8b: Iterating over the tuple – **for** – loops

1.1.8 ENUMERATING OVER LISTS



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> lists = ["l", "i", "s", "t"]
>>> for (i, x) in enumerate(lists):
...     print(i, x)
...
0 l
1 i
2 s
3 t
>>> for i in range(len(lists)):
...     print(i, lists[i])
...
0 l
1 i
2 s
3 t
>>> 
```

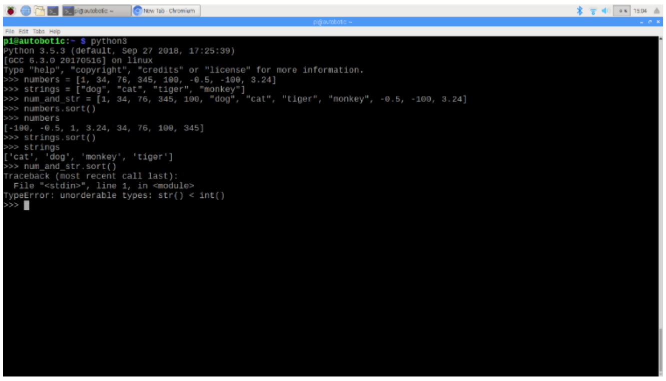
Figure 9a: Enumerate over the list – **for** – loops



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> tuples = ("t", "u", "p", "l", "e")
>>> for (i, x) in enumerate(tuples):
...     print(i, x)
...
0 t
1 u
2 p
3 l
4 e
>>> for i in range(len(tuples)):
...     print(i, tuples[i])
...
0 t
1 u
2 p
3 l
4 e
>>> 
```

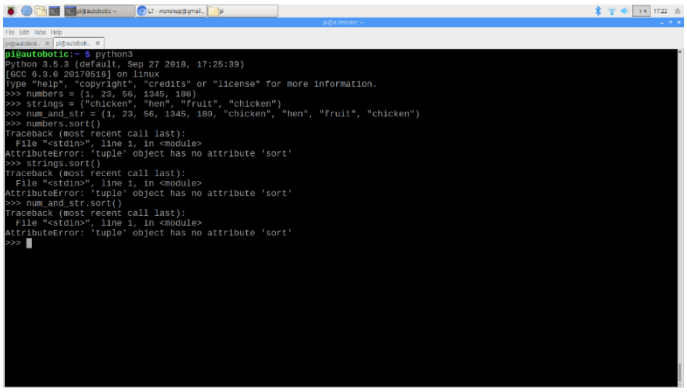
Figure 9b: Figure 9a: Enumerate over the tuple – **for** – loops

1.1.9 SORT



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = [1, 34, 76, 345, 100, -0.5, -100, 3.24]
>>> strings = ["dog", "cat", "tiger", "monkey"]
>>> num_and_str = [1, 34, 76, 345, 100, "dog", "cat", "tiger", "monkey", -0.5, -100, 3.24]
>>> numbers.sort()
>>> numbers
[-100, -0.5, 1, 3.24, 34, 76, 100, 345]
>>> strings.sort()
>>> strings
['cat', 'dog', 'monkey', 'tiger']
>>> num_and_str.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: str() < int()
>>> 
```

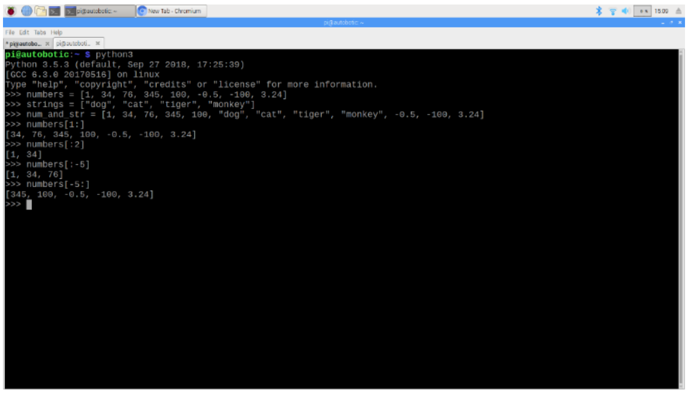
Figure 10a: Sort a lists using **sort()** function



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = (1, 23, 56, 1345, 180)
>>> strings = ("chicken", "hen", "fruit", "chicken")
>>> num_and_str = (1, 23, 56, 1345, 180, "chicken", "hen", "fruit", "chicken")
>>> numbers.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'sort'
>>> strings.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'sort'
>>> num_and_str.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'sort'
>>> 
```

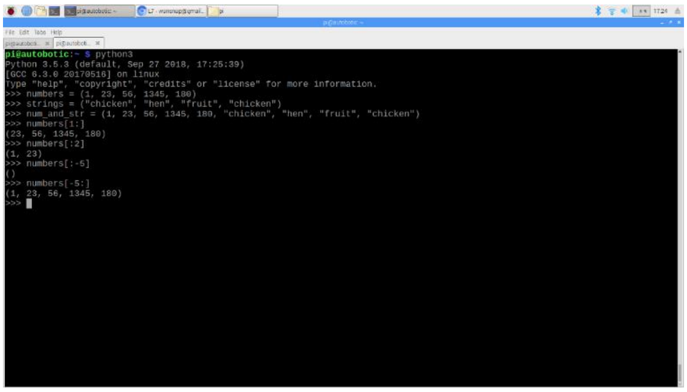
Figure 10b: cannot sort tuple – there is now better way to change the tuple arrangement (add/remove/etc) – re-initialize new item on tuple only

1.1.10 CUTTING UP



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = [1, 34, 76, 345, 100, -0.5, -100, 3.24]
>>> strings = ["dog", "cat", "tiger", "monkey"]
>>> num_and_str = [1, 34, 76, 345, 100, "dog", "cat", "tiger", "monkey", -0.5, -100, 3.24]
>>> numbers[1:]
[34, 76, 345, 100, -0.5, -100, 3.24]
>>> numbers[:2]
[1, 34]
>>> numbers[:-5]
[1, 34, 76]
>>> numbers[-5:]
[345, 100, -0.5, -100, 3.24]
>>> 
```

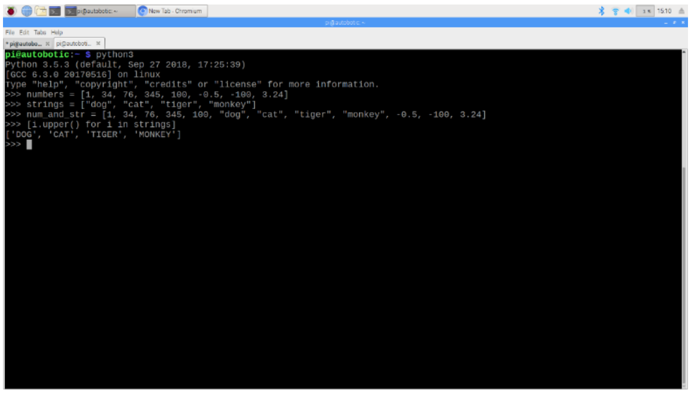
Figure 11a: Select required item only in list using **[:]**



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = (1, 23, 56, 1345, 180)
>>> strings = ("chicken", "hen", "fruit", "chicken")
>>> num_and_str = (1, 23, 56, 1345, 180, "chicken", "hen", "fruit", "chicken")
>>> numbers[1:]
(23, 56, 1345, 180)
>>> numbers[:2]
(1, 23)
>>> numbers[:-5]
()
>>> numbers[-5:]
(1, 23, 56, 1345, 180)
>>> 
```

Figure 11b: Select required item only in tuple using **[:]**

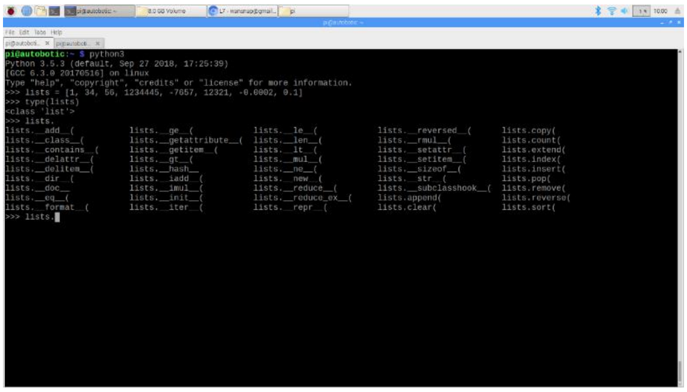
1.1.11 APPLYING FUNCTION TO A LISTS



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> numbers = [1, 34, 76, 345, 100, -0.5, -100, 3.24]
>>> strings = ["dog", "cat", "tiger", "monkey"]
>>> num_and_str = [1, 34, 76, 345, 100, "dog", "cat", "tiger", "monkey", -0.5, -100, 3.24]
>>> [i.upper() for i in strings]
['DOG', 'CAT', 'TIGER', 'MONKEY']
>>> 
```

Figure 12: Checking how many elements there are in a list

1.1.12 MORE OPERATION ON LIST AND TUPLE

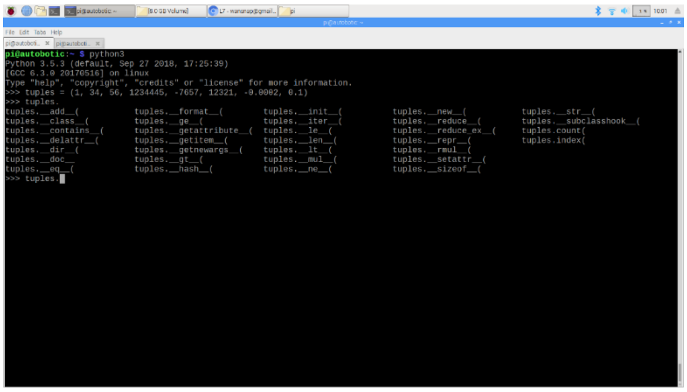


```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> lists = [1, 34, 56, 1234445, -7657, 12321, -0.0002, 0.1]
>>> type(lists)
<class 'list'>
>>> lists.
lists.__add__(          lists.__ge__(           lists.__le__(           lists.__reversed__(     lists.copy(
lists.__class__(        lists.__getattribute__( lists.__len__(          lists.__rmul__(         lists.count(
lists.__contains__(     lists.__getitem__(      lists.__lt__(           lists.__setattr__(      lists.extend(
lists.__delattr__(      lists.__gt__(           lists.__mul__(          lists.__setitem__(      lists.index(
lists.__delitem__(      lists.__hash__          lists.__ne__(           lists.__sizeof__(       lists.insert(
lists.__dir__(          lists.__iadd__(         lists.__new__(          lists.__str__(          lists.pop(
lists.__doc__           lists.__imul__(         lists.__reduce__(       lists.__subclasshook__( lists.remove(
lists.__eq__(           lists.__init__(         lists.__reduce_ex__(    lists.append(           lists.reverse(
lists.__format__(       lists.__iter__(         lists.__repr__(         lists.clear(            lists.sort(
>>> lists.
```

Figure 13: Using **tab** to find out more list operation

Operator	Meaning
list.append(item)	Add item to the end of the list
list.extend(new_list)	Join new_list to the end of list
list.pop(x)	Return and remove x-th item
list.insert(x, item)	Insert item at x-th position
list.sort()	Sort the list
list.index(item)	Return the position of the first occurrence of item in list
list.count(item)	Count how many times item appear in list
list.remove(item)	Remove the first occurrence of item in list

Table 1: Available list operation



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> tuples = (1, 34, 56, 1234445, -7657, 12321, -0.0002, 0.1)
>>> tuples.
tuples.__add__(          tuples.__format__(       tuples.__init__(         tuples.__new__(          tuples.__str__(
tuples.__class__(        tuples.__ge__(           tuples.__iter__(         tuples.__reduce__(       tuples.__subclasshook__(
tuples.__contains__(     tuples.__getattribute__( tuples.__le__(           tuples.__reduce_ex__(    tuples.count(
tuples.__delattr__(      tuples.__getitem__(      tuples.__len__(          tuples.__repr__(         tuples.index(
tuples.__dir__(          tuples.__getnewargs__(   tuples.__lt__(           tuples.__rmul__(
tuples.__doc__           tuples.__gt__(           tuples.__mul__(          tuples.__setattr__(
tuples.__eq__(           tuples.__hash__(         tuples.__ne__(           tuples.__sizeof__(
>>> tuples.
```

Figure 14: Using **tab** to find out more tuple operation

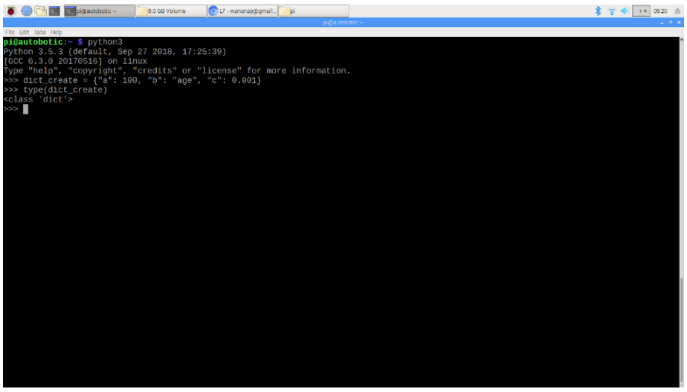
Operator	Meaning
tuple.count(item)	Count how many times item appear in list
tuple.remove(item)	Remove the first occurrence of item in list

Table 2: Available list operation

1.2 DICTIONARIES AND SETS

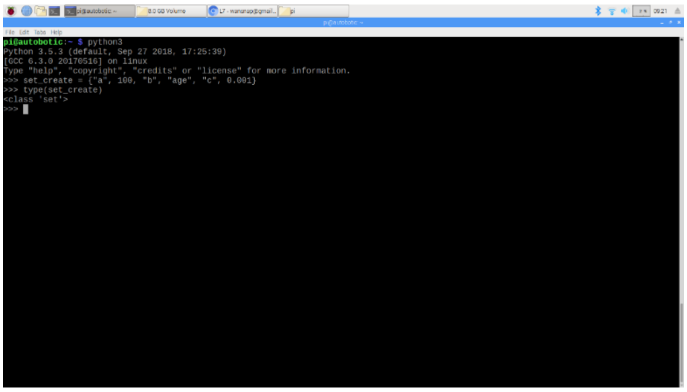
Dictionaries – key/value pairs – is a lookup table where you associate values with keys. Dictionaries are an alternative to lists for storing collections of data, but they are organized very differently – unordered. Another unordered sequence, we may use – sets; values only. Both created with curly braces.

1.2.1 CREATING A DICTIONARY AND SET



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> dict_create = {"a": 100, "b": "age", "c": 0.001}
>>> type(dict_create)
<class 'dict'>
>>> 
```

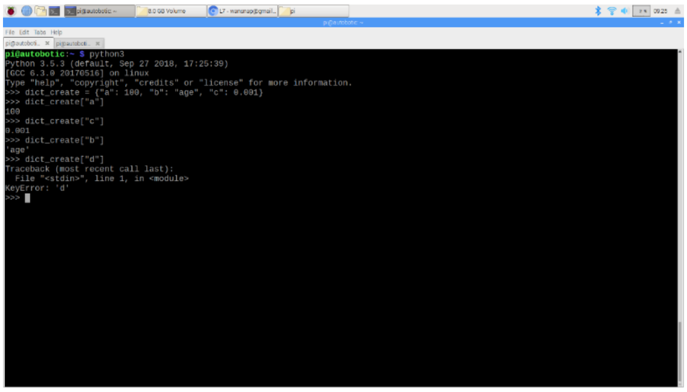
Figure 15a: Create dictionary – curly braces



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> set_create = {"a", 100, "b", "age", "c", 0.001}
>>> type(set_create)
<class 'set'>
>>> 
```

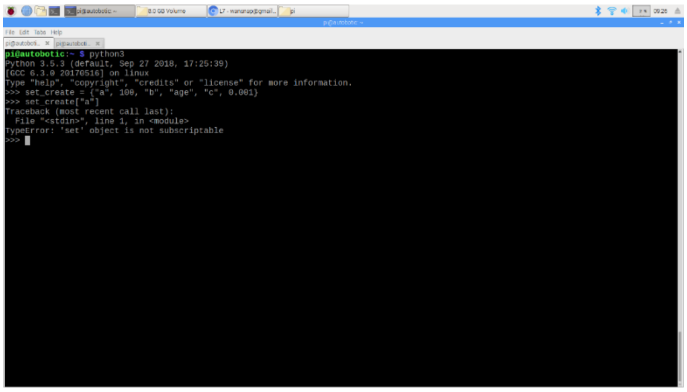
Figure 15b: Create sets – curly braces

1.2.2 ACCESSING A DICTIONARY AND SETS



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> dict_create = {"a": 100, "b": "age", "c": 0.001}
>>> dict_create["a"]
100
>>> dict_create["c"]
0.001
>>> dict_create["b"]
'age'
>>> dict_create["d"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'd'
>>> 
```

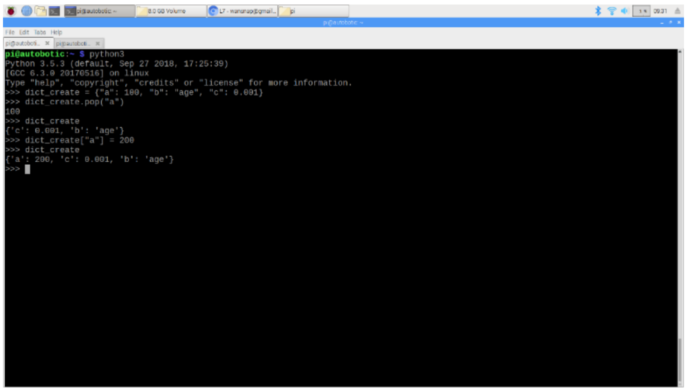
Figure 16a: Accessing dictionary – [] and “key”



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> set_create = {"a", 100, "b", "age", "c", 0.001}
>>> set_create["a"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable
>>> 
```

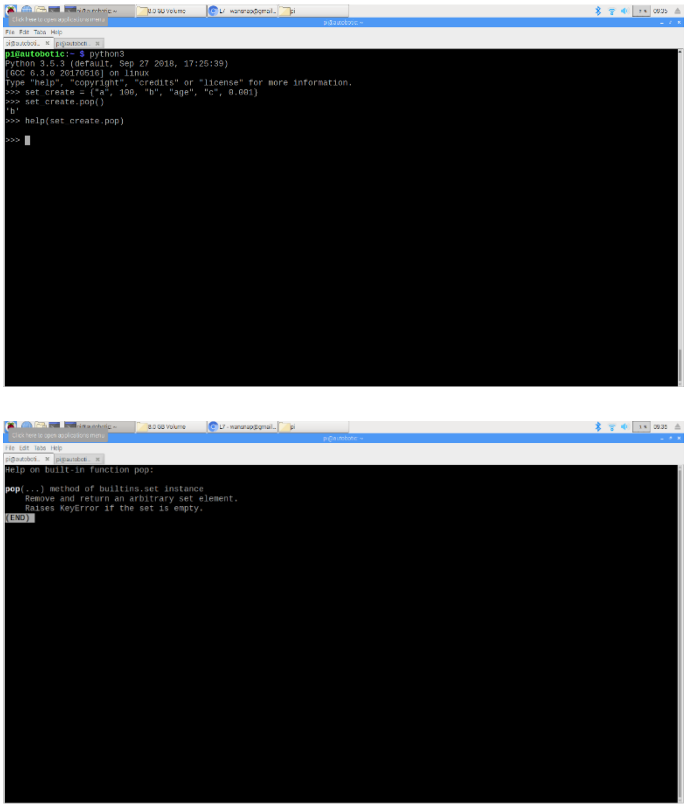
Figure 16b: Accessing the sets – not available

1.2.3 REMOVING/ADDING THINGS FROM A DICTIONARY



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> dict_create = {"a": 100, "b": "age", "c": 0.001}
>>> dict_create.pop("a")
100
>>> dict_create
{'c': 0.001, 'b': 'age'}
>>> dict_create["a"] = 200
>>> dict_create
{'a': 200, 'c': 0.001, 'b': 'age'}
>>> 
```

Figure 17a: Adding items/things to dictionary

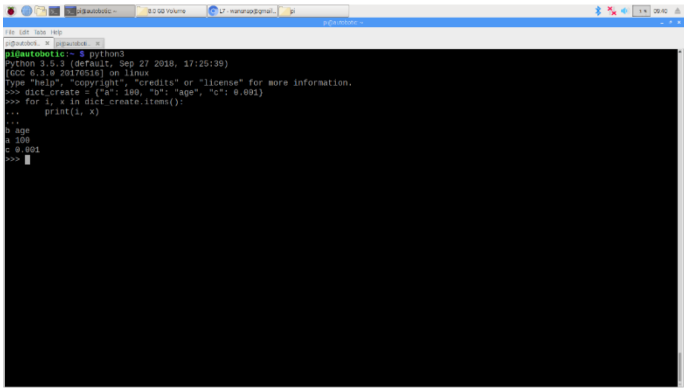


```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> set_create = {"a", 100, "b", "age", "c", 0.001}
>>> set_create.pop()
'b'
>>> help(set_create.pop)

>>> 
```

```
Help on built-in function pop:

pop(...) method of builtins.set instance
    Remove and return an arbitrary set element.
    Raises KeyError if the set is empty.
(END)
```

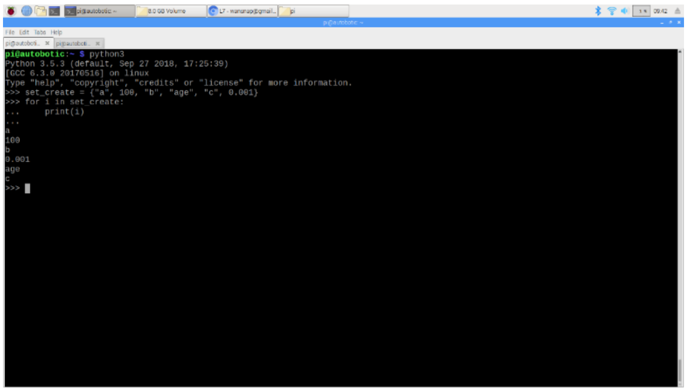
Figure 17b: Adding/removing item/things in sets not available – like tuple too.

1.2.4 ITERATING OVER DICTIONARY



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> dict_create = {"a": 100, "b": "age", "c": 0.001}
>>> for i, x in dict_create.items():
...     print(i, x)
...
b age
a 100
c 0.001
>>> 
```

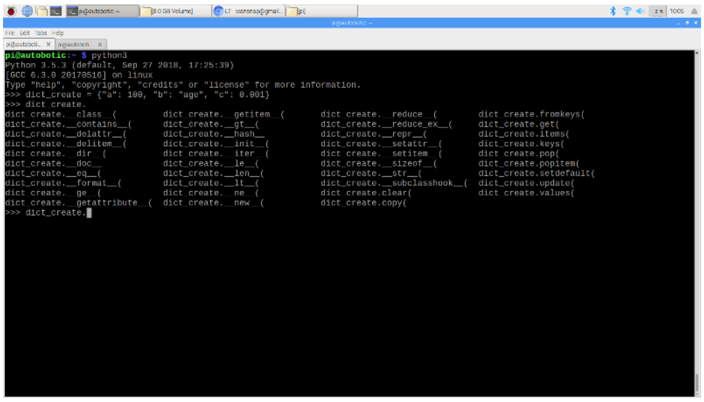
Figure 18a: Using – **for** – loops to iterate in dictionary



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> set_create = {"a", 100, "b", "age", "c", 0.001}
>>> for i in set_create:
...     print(i)
...
a
100
b
0.001
age
c
>>> 
```

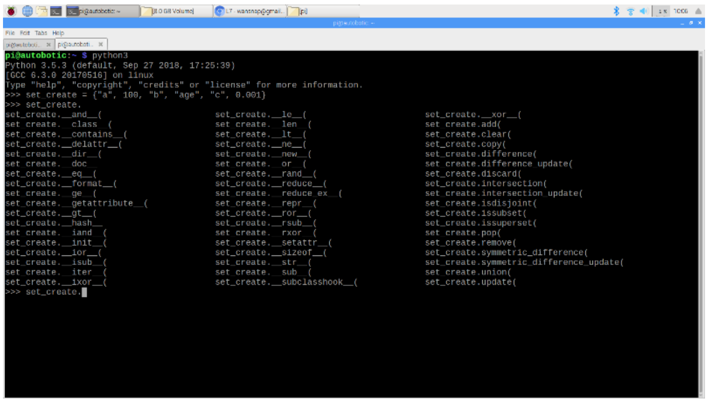
Figure 18b: Using – **for** – loops to iterate in sets

1.2.5 MORE DICTIONARY AND SET OPERATIONS



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> dict_create = {"a": 100, "b": "age", "c": 0.001}
>>> dict_create.
dict_create.__class__(          dict_create.__getitem__(        dict_create.__reduce__(         dict_create.fromkeys(
dict_create.__contains__(       dict_create.__gt__(             dict_create.__reduce_ex__(      dict_create.get(
dict_create.__delattr__(        dict_create.__hash__            dict_create.__repr__(           dict_create.items(
dict_create.__delitem__(        dict_create.__init__(           dict_create.__setattr__(        dict_create.keys(
dict_create.__dir__(            dict_create.__iter__(           dict_create.__setitem__(        dict_create.pop(
dict_create.__doc__             dict_create.__le__(             dict_create.__sizeof__(         dict_create.popitem(
dict_create.__eq__(             dict_create.__len__(            dict_create.__str__(            dict_create.setdefault(
dict_create.__format__(         dict_create.__lt__(             dict_create.__subclasshook__(   dict_create.update(
dict_create.__ge__(             dict_create.__ne__(             dict_create.clear(              dict_create.values(
dict_create.__getattribute__(   dict_create.__new__(            dict_create.copy(
>>> dict_create.
```

Figure 19a: Using **tab** to find out more dictionary operation



```
pi@autobotic:~ $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> set_create = {"a", 100, "b", "age", "c", 0.001}
>>> set_create.
set_create.__and__(                      set_create.__le__(                       set_create.__xor__(
set_create.__class__(                    set_create.__len__(                      set_create.add(
set_create.__contains__(                 set_create.__lt__(                       set_create.clear(
set_create.__delattr__(                  set_create.__ne__(                       set_create.copy(
set_create.__dir__(                      set_create.__new__(                      set_create.difference(
set_create.__doc__                       set_create.__or__(                       set_create.difference_update(
set_create.__eq__(                       set_create.__rand__(                     set_create.discard(
set_create.__format__(                   set_create.__reduce__(                   set_create.intersection(
set_create.__ge__(                       set_create.__reduce_ex__(                set_create.intersection_update(
set_create.__getattribute__(             set_create.__repr__(                     set_create.isdisjoint(
set_create.__gt__(                       set_create.__ror__(                      set_create.issubset(
set_create.__hash__                      set_create.__rsub__(                     set_create.issuperset(
set_create.__iand__(                     set_create.__rxor__(                     set_create.pop(
set_create.__init__(                     set_create.__setattr__(                  set_create.remove(
set_create.__ior__(                      set_create.__sizeof__(                   set_create.symmetric_difference(
set_create.__isub__(                     set_create.__str__(                      set_create.symmetric_difference_update(
set_create.__iter__(                     set_create.__sub__(                      set_create.union(
set_create.__ixor__(                     set_create.__subclasshook__(             set_create.update(
>>> set_create.
```

Figure 19b: Using **tab** to find out more set operation

Operator	Meaning
set1 & set2	Return the items that are in both set
set1 | set2	Combine an item in both set
set1 - set2	The items set 1 aren't in set 2
set1 ^ set2	The item that are in set 1 or set 2, but not both

Table 3: Available set operation

2 CHALLENGE

Let's play Rock, Paper, Scissors! For this challenge, try to create the classic game. Rock, Paper, Scissors is played with your hands. Each person simultaneously makes one of three shapes with their hand: the shape of a rock, a piece of paper, or a pair of scissors. If two people make the same shape, it's a tie. The three game shapes interact with each other like this:

1. Rock beats scissors.
2. Paper beats rock.
3. Scissors beats paper.

Let's plan how to attack this challenge. Here are some of the key elements:

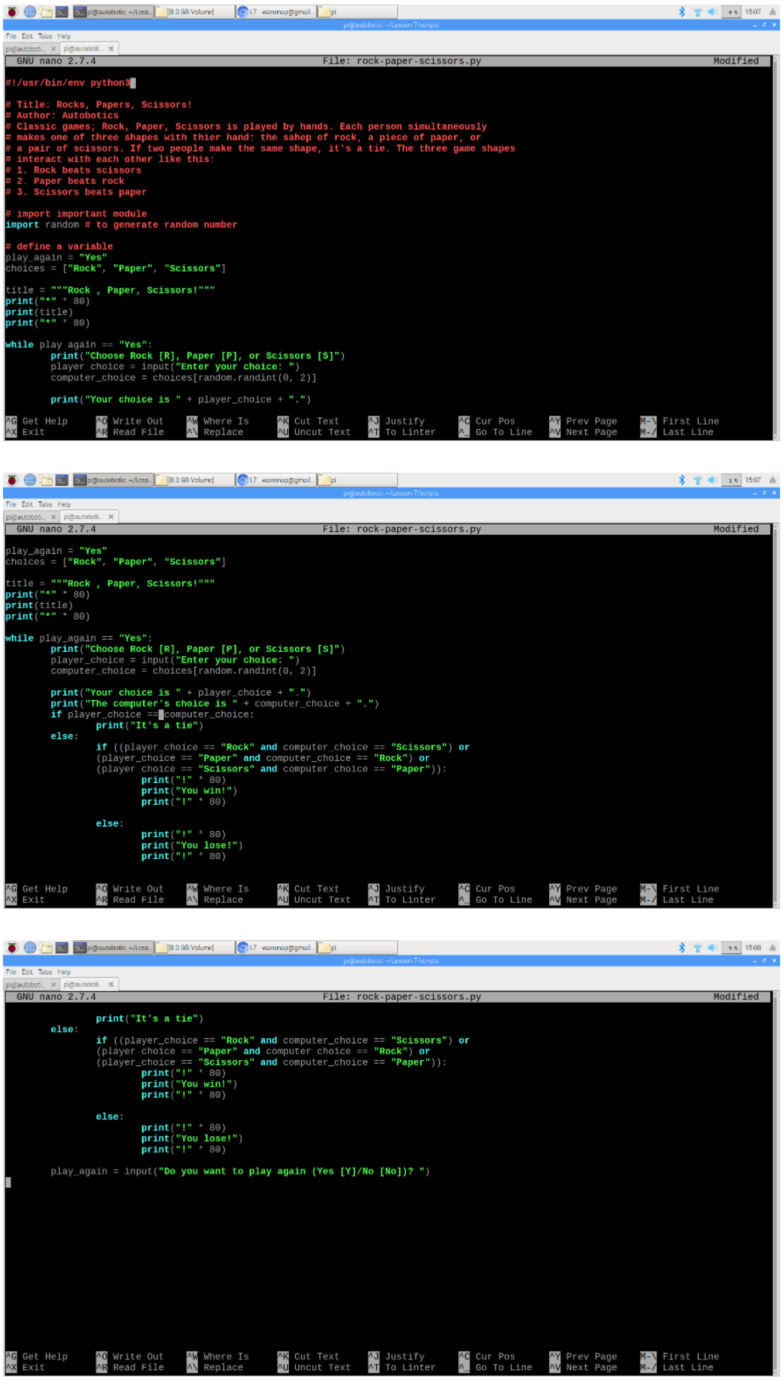
Use a while loop to repeatedly ask the player to choose rock, paper, or scissors.

1. Create a list of choices:
    a. choices = ["Rock","Paper","Scissors"]
2. Use the random library to have the computer randomly choose among the three choices ("Rock", "Paper", and "Scissors")
3. Remember, randint selects a random integer. You can select and store the random choice in a variable:
    a. computer_choice = choices[random.randint(0,2)]
4. You can select different items in the list by using a number representing where the item is in the list. This number is called a list index. In this case, there are three items in the list. The first item has an index of 0, the second item has an index of 1, and the third item has an

index of 2. To display the second item in the list, you write print(choices[1]); the code displays "Paper" on the screen.

5. Use an if statement to compare the player's choice to the computer's choice and let the player know who won.
6. Ask the player if they want to play again. If so, the loop should repeat; if not, the game should end.

2.1 FINISHED CODE: PUTTING ALL TOGETHER



```
  GNU nano 2.7.4                 File: rock-paper-scissors.py                  Modified

#!/usr/bin/env python3

# Title: Rocks, Papers, Scissors!
# Author: Autobotics
# Classic games; Rock, Paper, Scissors is played by hands. Each person simultaneously
# makes one of three shapes with thier hand: the sahep of rock, a piece of paper, or
# a pair of scissors. If two people make the same shape, it's a tie. The three game shapes
# interact with each other like this:
# 1. Rock beats scissors
# 2. Paper beats rock
# 3. Scissors beats paper

# import important module
import random # to generate random number

# define a variable
play_again = "Yes"
choices = ["Rock", "Paper", "Scissors"]

title = """Rock , Paper, Scissors!"""
print("*" * 80)
print(title)
print("*" * 80)

while play_again == "Yes":
        print("Choose Rock [R], Paper [P], or Scissors [S]")
        player_choice = input("Enter your choice: ")
        computer_choice = choices[random.randint(0, 2)]

        print("Your choice is " + player_choice + ".")
```

```
  GNU nano 2.7.4                 File: rock-paper-scissors.py                  Modified

play_again = "Yes"
choices = ["Rock", "Paper", "Scissors"]

title = """Rock , Paper, Scissors!"""
print("*" * 80)
print(title)
print("*" * 80)

while play_again == "Yes":
        print("Choose Rock [R], Paper [P], or Scissors [S]")
        player_choice = input("Enter your choice: ")
        computer_choice = choices[random.randint(0, 2)]

        print("Your choice is " + player_choice + ".")
        print("The computer's choice is " + computer_choice + ".")
        if player_choice == computer_choice:
                print("It's a tie")
        else:
                if ((player_choice == "Rock" and computer_choice == "Scissors") or
                (player_choice == "Paper" and computer_choice == "Rock") or
                (player_choice == "Scissors" and computer_choice == "Paper")):
                        print("!" * 80)
                        print("You win!")
                        print("!" * 80)

                else:
                        print("!" * 80)
                        print("You lose!")
                        print("!" * 80)
```

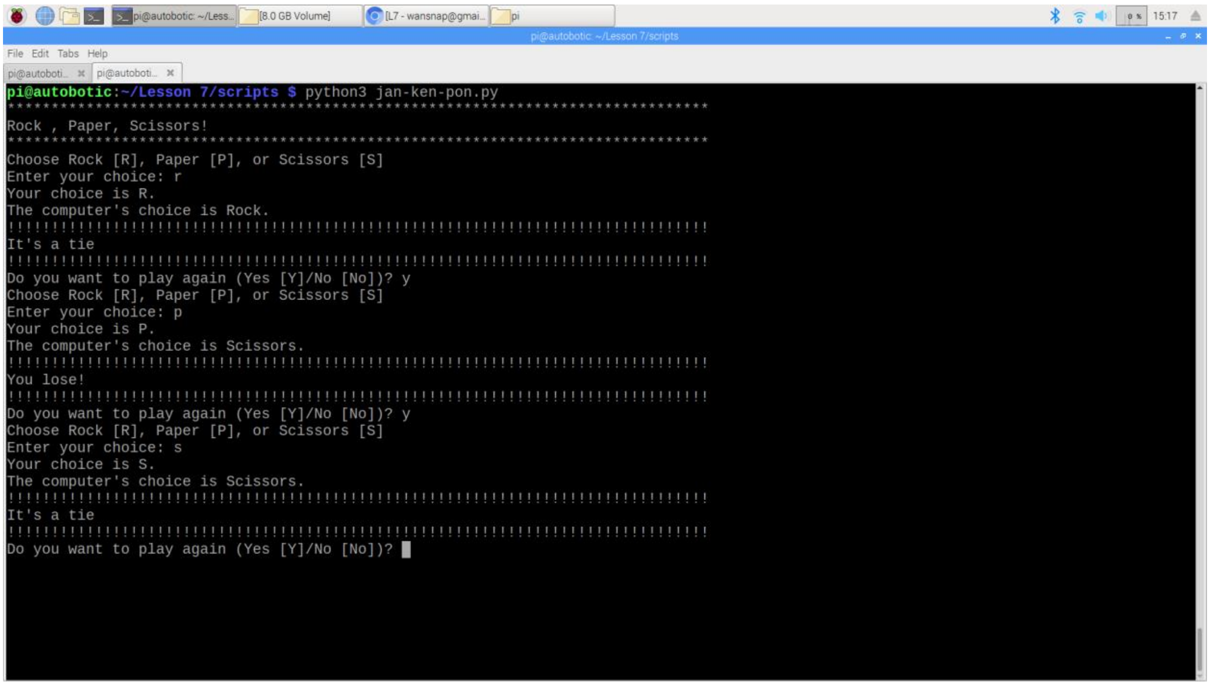
```
  GNU nano 2.7.4                 File: rock-paper-scissors.py                  Modified

                print("It's a tie")
        else:
                if ((player_choice == "Rock" and computer_choice == "Scissors") or
                (player_choice == "Paper" and computer_choice == "Rock") or
                (player_choice == "Scissors" and computer_choice == "Paper")):
                        print("!" * 80)
                        print("You win!")
                        print("!" * 80)

                else:
                        print("!" * 80)
                        print("You lose!")
                        print("!" * 80)

        play_again = input("Do you want to play again (Yes [Y]/No [No])? ")
```

Figure 20: Using **nano** to write a rock, paper, or scissor game – text based

```
#!/usr/bin/env python3

# Title: Rocks, Papers, Scissors!
# Author: Autobotics
# Classic games; Rock, Paper, Scissors is played by hands. Each person simultaneously
# makes one of three shapes with thier hand: the sahep of rock, a piece of paper, or
# a pair of scissors. If two people make the same shape, it's a tie. The three game shapes
# interact with each other like this:
# 1. Rock beats scissors
# 2. Paper beats rock
# 3. Scissors beats paper

# import important module
import random # to generate random number

# define a variable
play_again = "Yes"
choices = ["Rock", "Paper", "Scissors"]

title = """Rock , Paper, Scissors!"""
print("*" * 80)
print(title)
print("*" * 80)

while play_again == "Yes":
    print("Choose Rock [R], Paper [P], or Scissors [S]")
    player_choice = input("Enter your choice: ")
    computer_choice = choices[random.randint(0, 2)]

    print("Your choiceis " + player_choice + ".")
    print("The computer's choice is " + computer_choice + ".")
    if player_choice == computer_choice:
        print("It's a tie")
```

```
else:
    if ((player_choice == "Rock" and computer_choice == "Scissors") or
    (player_choice == "Paper" and computer_choice == "Rock") or
    (player_choice == "Scissors" and computer_choice == "Paper")):
        print("!" * 80)
        print("You win!")
        print("!" * 80)

    else:
        print("!" * 80)
        print("You lose!")
        print("!" * 80)

play_again = input("Do you want to play again (Yes [Y]/No [No])? ")
```

2.1.1 PLAYING THE GAME



```
pi@autobotic:~/Lesson 7/scripts $ python3 jan-ken-pon.py
********************************************************************************
Rock , Paper, Scissors!
********************************************************************************
Choose Rock [R], Paper [P], or Scissors [S]
Enter your choice: r
Your choice is R.
The computer's choice is Rock.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
It's a tie
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Do you want to play again (Yes [Y]/No [No])? y
Choose Rock [R], Paper [P], or Scissors [S]
Enter your choice: p
Your choice is P.
The computer's choice is Scissors.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
You lose!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Do you want to play again (Yes [Y]/No [No])? y
Choose Rock [R], Paper [P], or Scissors [S]
Enter your choice: s
Your choice is S.
The computer's choice is Scissors.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
It's a tie
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Do you want to play again (Yes [Y]/No [No])? █
```

Figure 21: Rock, Paper, or Scissors!