**Lab: List, Tuples and Dictionaries**

Work with your group – GO THROUGH THE POWERPOINT – try the activities.

**Watch the parentheses, braces and brackets. They make all the difference.**

**Problem 0**

What is a Method? A function inside of a function.

What is an immutable data structure? A data structure you can’t change. Name 4: int, string, tuple, bool

What is a mutable data structure? A data structure you can change. Name 2: list, dictionary

**Lists**

What defines a list? A changeable data structure that holds a series of items.

Are there list methods? Yes

In IDLE enter

>>> dir(list)

How do these differ from what you found for strings? >>>dir(str) if you need a reminder.

There are more methods for str than list and the methods for lists can change a list vs string which can only be replaced/reassigned.

Then ask Python for help to learn about the method\_descriptors. Paste the information on the method\_descriptors: 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', and 'sort'. Fill out the table. Again as a reminder >>> help(list.pop)

Use colors for examples:

colors = [red, orange, yellow, green]

|  |  |  |
| --- | --- | --- |
| Method (format) | Description | Example (copy shell) |
| list.append([index]) | append object to end | >>> colors.append('blue')  >>> ['red', 'orange', 'yellow', 'green', 'blue'] |
| List.clear() | Removes all items from list | >>> Colors.clear()  >>> [] |
| List.copy() | A shallow copy of list | Colors.copy()  >>> ['red', 'orange', 'yellow', 'green', 'blue', 'perrywinkle', 'purple'] |
| List.count(value) | return number of occurrences of value | Colors.count(‘red’)  >>> 1 |
| List.extend(iterable) | Extend list by appending elements from the iterable | >>> colors2 = ['perrywinkle', 'purple']  >>> colors.extend(colors2)  >>> print(colors)  ['red', 'orange', 'yellow', 'green', 'blue', 'perrywinkle', 'purple'] |
| List.index(value, [start, [stop]]) | Return first index of the value. Raises ValueError if the value is not present. | >>> colors.index('blue')  4 |
| List.insert(index, object) | Insert object before index | >>> colors.insert(3, 'teal')  >>> print(colors)  ['red', 'orange', 'yellow', 'teal', 'green', 'blue', 'perrywinkle', 'purple'] |
| List.pop([index]) | remove and return item at index (default last).  Raises IndexError if list is empty or index is out of range. | >>> print(colors.pop(4))  green  >>> print(colors)  ['red', 'orange', 'yellow', 'teal', 'blue', 'perrywinkle', 'purple'] |
| List.remove(value) | remove first occurrence of value.  Raises ValueError if the value is not present. | >>> colors.remove('orange')  >>> print(colors)  ['red', 'yellow', 'teal', 'blue', 'perrywinkle', 'purple'] |
| List.reverse() | reverse \*IN PLACE\* | >>> colors.reverse()  >>> print(colors)  ['purple', 'perrywinkle', 'blue', 'teal', 'yellow', 'red'] |
| List.sort(key=none, reverse=false) | stable sort \*IN PLACE\* | >>> colors.sort()  >>> print(colors)  ['blue', 'perrywinkle', 'purple', 'red', 'teal', 'yellow'] |

**Tuples**

Tuples are immutable sequences: they cannot be modified. Tuples and lists have much in common, but lists are mutable sequences: they can be modified.

Tuples use parentheses instead of square brackets:

lst = ['a', 3, -0.2]

tup = ('a', 3, -0.2)

Once created, items in lists and tuples are accessed using the same notation:

>>> lst[0]

'a'

>>> tup[0]

'a'

Slicing can be used with both:

>>> lst[:2]

['a', 3]

>>> tup[:2]

('a', 3)

Tuples cannot be modified:

>>> tup[0] = 'b'

TypeError: 'tuple' object does not support item assignment

Tuples have fewer methods than lists. In fact, the only regular methods are count and index:

**Find:**

>>> dir(list)

>>> dir(tuple)

The rest of the list methods are not available for tuple because they modify the object, and tuples, being immutable, cannot be modified.

A for can be used to iterate over the values in a tuple:

>>> tup = ('a', 3, -0.2)

>>> for item in tup:

print(item)

a

3

-0.2

A tuple can be passed as an argument to the built-in function len:

>>> len(tup)

3

It is also possible to iterate over the indices of a tuple:

>>> for i in range(len(tup)):

print(tup[i])

***What is the output here?***

**a**

**3**

**-0.2**

**In your own words:**

What is a Tuple? What defines a Tuple? A series of objects that you can’t change, with less function.

Are there Tuple methods? Yes

Is a Tuple an immutable data structure or a mutable data structure? immutable

Compare what you found above for

>>>dir(tuple) only has index() and count() for methods.

>>>dir(list) has all of the methods in the above table.

Similarities? Differences? Why? Index() and count() are similar in both because they both have indices with values stored in them. Lists have all of those other methods because it is mutable.

Then ask Python for help to learn about the method\_descriptors. Paste the information on the method\_descriptors:

Fill out the table.

How do you **change** the color list so it is a tuple? tuple(colors)

For Lists we used colors for examples:

colors = [red, orange, yellow, green]

|  |  |  |
| --- | --- | --- |
| Method (format) | Description | Example (copy shell) |
| Tuple.count(value, [start, [stop]]) | return first index of value.  Raises ValueError if the value is not present. | >>> colors\_tup.index('red')  0 |
| Tuple.count(value) | return number of occurrences of value | >>> colors\_tup.count('red')  1 |

**Dictionaries**

Another way to store collections of data is using Python's dictionary type: dict.

The general form of a dictionary is:

{key1: value1, key2: value2, ..., keyN: valueN}

Keys must be unique. Values may be duplicated. For example:

asn\_to\_grade = {'A1': 80, 'A2': 90, 'A3': 90}

In the example above, the keys are unique: 'A1', 'A2' and 'A3'. The values are not unique: 80, 90 and 90.

Dictionaries are mutable: they can be modified. There are a series of operations and methods you can apply to dictionaries which are outlined below.

What defines a dictionary? A dictionary is a series of specific keys stored with their assigned value.

Are there dictionary methods? yes

In IDLE enter

>>> dir(dict)

How do these differ from what you found for lists and tuples? >>>dir(list) if you need a reminder.

Then ask Python for help to learn about the method\_descriptors. Paste the information on the method\_descriptors: 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values'. Fill out the table. Again as a reminder >>> help(dict.clear)

Use fruitColor for examples:

>>> fruitColor = {'cherry' : 'red', 'orange' : 'orange', 'banana': 'yellow', 'apple' : 'green'}

|  |  |  |
| --- | --- | --- |
| **Method (format)** | **Description** | **Example (copy shell)** |
| dict.clear() | Remove all items from dict. | >>> fruitColor.clear()  >>> print(fruitColor)  {} |
| Dict.copy() | A shallow copy of dict | >>> fruitColor.copy()  {'cherry': 'red', 'banana': 'yellow', 'apple': 'green', 'orange': 'orange'} |
| Dict.fromkeys(iterable, value=none, /) | Returns a new dict with keys from iterable and values equal to value. | >>> fruit.fromkeys(fruitColor)  {'cherry': None, 'banana': None, 'apple': None, 'orange': None} |
| Dict.get(k[,d]) | D[k] if k in D, else d. d defaults to None. | >>> fruitColor.get('apple')  'green' |
| Dict.items() | a set-like object providing a view on D's items | >>> fruitColor.items()  dict\_items([('cherry', 'red'), ('banana', 'yellow'), ('apple', 'green'), ('orange', 'orange')]) |
| Dict.keys() | a set-like object providing a view on D's keys | >>> fruitColor.keys()  dict\_keys(['cherry', 'banana', 'apple', 'orange']) |
| Dict.pop(k[,d]) | v, remove specified key and return the corresponding value.  If key is not found, d is returned if given, otherwise KeyError is raised | >>> fruitColor.pop('apple')  'green' |
| Dict.popitem() | remove and return some (key, value) pair as a  2-tuple; but raise KeyError if D is empty. | >>> fruitColor.popitem()  ('cherry', 'red') |
| Dict.setdefault(k[,d]) | D.get(k,d), also set D[k]=d if k not in D | >>> fruitColor.setdefault('banana')  'yellow' |
| Dict.update([E, ]\*\*F) | Update D from dict/iterable E and F.  If E is present and has a .keys() method, then does: for k in E: D[k] = E[k]  If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v  In either case, this is followed by: for k in F: D[k] = F[k] | >>> fruitColor.update(fruitColor) |
| Dict.values() | An object providing a view on D’s values | dict\_values(['yellow', 'green', 'orange']) |
| **Operation** | **Description** | **Example** |
| object **in** dict | Looks in dict for object and returns true if found, or false if not found | >>> ‘banana’ in fruitColor  True |
| len(dict) | Returns how many items (key/value pairs) are in dict | >>> len(fruitColor)  4 |
| del dict[key] | Deletes the specified key/value pair from dict | >>> del fruitColor['cherry']  >>> fruitColor  {'banana': 'yellow', 'apple': 'green', 'orange': 'orange'} |
| dict[key] = value | Adds a new key to dict and assigns value to pair it | >>> fruitColor['lime'] = 'green' |

for fruit in fruitColor:

print(fruit)

Did it print in the order you entered? Yes

The for-loop above printed out the keys of the dictionary. It is also possible to print out the values:

for fruit in fruitColor:

print(fruitColor[fruit])

Finally, both the keys are values can be printed:

for fruit in fruitColor:

print(fruit, fruitColor[fruit])

### Empty Dictionaries

A dictionary can be empty. For example:

d = {}

### Heterogeneous Dictionaries

A dictionary can have keys of different types. For example, one key can be of type int and another of type str:

d = {'apple': 1, 3: 4}

### Immutable Keys

The keys of a dictionary must be immutable. Therefore, lists, dictionary and other mutable types cannot be used as keys. The following results in an error:

d[[1, 2]] = 'banana'

Since lists are mutable, they cannot be keys. Instead, to use a sequence as a key, type tuple can be used:

d[(1, 2)] = 'banana'

**Finale:**

1. List 4 immutable data structures? Strings, tuples, integers, boolean
2. List 2 mutable data structures? Lists, dictionaries
3. The symbol that defines a list is: []
4. The symbol that defines a tuple is: ()
5. The symbol that defines a dictionary is: {}