Introduction to Software Systems (CS6.201)

Summer 2021, IIIT Hyderabad

01 July, Thursday (Lecture 11) – Python 2

Taught by Abhinav Gupta

Modules

Python has a set of standard libraries, which implement several useful functions. A library's functions can be used in code by importing it. For example, we can use the `random` library to generate 10 random integers between 1 and 10 (inclusive):

```
import random
for i in range(10):
    print(random.randint(1,10))
```

The `.` indicates that the `randint` function is part of the `random` library.

On the other hand, if we simply import the functions from the library, we needn't use this syntax. For instance

```
from random import randint
for i in range(10):
    print(randint(1,10))
```

Lists, Tuples, Sets and Dictionaries

Lists

Lists are denoted in square brackets with comma-separated values, as in

```
marks = [77,88,90,100]
```

They can be indexed into using square brackets; thus `mark[0]` returns 77. We can also use negative indexing to index from the end: `mark[-2]` returns 90.

Lists can be sliced with colon notation; for example, `marks[1:-1]` returns `[88,90]`. Note that the last index is excluded.

Lists are mutable; thus their indices can act as lvalues as well. The following statement:

```
marks[0] = 71
```

does not throw an error.

A entry of a list can be deleted as well; for instance, `del marks[0]` causes `marks` to be `[88,90,100]`.

One can also assign each element of a list to a variable:

```
marks1, marks2, marks3 = marks
```

Now `marks1` has 88, `marks2` 90 and `marks3` 100.

We can find the index of a certain value using the `index` method: `marks.index(100)` returns 2.

Lists can be appended to using the `append` method: `marks.append(97)` causes `marks` to be `[88,90,100,97]`.

One can insert at any index using the `insert` method. `marks.insert(0,85)` causes marks to be `[85,88,90,100,97]`.

Entries can be removed by value; `marks.remove(100)` causes `marks` to be `[85,88,90,97]`. It deletes the first occurrence of the value).

Lists can be sorted using the `sort` method, in either ascending or descending order: `marks.sort()` does not change `marks`; `marks.sort(reverse=True)` reverses it.

Note that variables that rely on list values store references to the values; thus when they are changed, the list is changed too and vice versa.

Tuples

Tuples are distinct from lists in that they are immutable. They are denoted by round brackets and comma-separated values.

```
t_marks = (77,80,90,95)
```

Being immutable, tuple values are not references, but actual values (unlike lists).

A singleton tuple must have a comma to distinguish it from a simple value; thus `(100,)` is a tuple but `(100)` is not.

Sets

Sets are enclosed in curly brackets and have comma-separated values. They are not allowed to have duplicates and the order of the values is not defined.

```
s_makrs = {77,77,88,90}
```

Now, `s_marks` may contain `{77,88,90}`.

Sets are mutable.

Dictionaries

Dictionaries are a mutable data type that stores a relation between keys and values. Their syntax is similar to sets:

```
mydict = {'name': 'Bani','age': 19}
```

Dictionaries are also unordered. Now the keys behave exactly like list indices; therefore `mydict['name']` returns `'Bani'`.

To iterate over key-value pairs, one must use the `items` method:

```
for key,val in mydict.items():
    print(key,val)
```

Similarly, one can iterate over only keys using the `keys` method and over only the values using the `values` method.

The `get` method allows us to supply a default return value for when the key does not exist; thus `mydict.get('name',0)` returns `'Bani'`, but `mydict.get('house',0)` returns 0.

Similarly, `setdefault` allows us to set a key's value if they key does not already exist.