Iterables and types

Monday, May 6, 2019 10:35 AM

There are many more kinds of values in Python.

- Lists: `[…]`
- Tuples: `(…)`
- Dicts: `{x:y, …}`

These differ in how they are handled and what they are intended to represent.

Pretty printing

- Consider
  ```
  from pprint import pprint
  pprint([1, 2, 3])
  ```
- This prints the structure in a way that indicates the structure.
- This prints:
  ```
  [1, 2, 3]
  ```
- The statement:
  ```
  from pprint import pprint
  ```
  instructs Python to load `pprint` from a library of useful functions.
- The import only has to be done once.

A list represents a list of items that can grow and shrink.

- Consider
  ```
  foo = []
  foo.append("cats")
  ```

```
foo.append("are")
foo.append("fun!")
from pprint import pprint
pprint(foo)
```

- This prints:

  ```
  ['cats', 'are', 'fun!']
  ```

- The function `pprint` is really powerful; it is capable of "pretty-printing" any value, regardless of structure. More about this later.

A `tuple` consists of a list of items that has a fixed size, where position indicates meaning of the item.

- Consider:

  ```
  n = ('cats', 10)
  m = ('dogs', 20)
  ```

  to represent that there are 10 cats and 20 dogs.
- After this:

  ```
  from pprint import pprint
  pprint(n)
  ```

  prints

  ```
  ('cats', 10)
  ```

A **dictionary** (or simply a '**dict**') consists of pairs, where the first is a key and the second is a value corresponding to that key.

- consider

  ```
  d = { 'cats':10, 'dogs':20 }
  ```

- After this:

  ```
  print(d['cats'])
  print(d['dogs'])
  ```

- prints
  ```
  10
  20
  ```

We've already studied how to iterate over lists.

```
for item in items:
    <do something with item>
```

How do we iterate over other structures?

Tuples

- Consider
  ```
  t = ('cats', 10)
  for d in t:
      print(d)
  ```
- This prints
  ```
  cats
  10
  ```

Dicts

- This is a bit counter-intuitive. Suppose we have the dict
  ```
  pets = {'cats': 10, 'dogs': 20}
  ```
- Then to print that, one might write:
  ```
  for k in pets:
      print(k)
      print(pets[k])
  ```
- This prints
  ```
  cats
  10
  ```

```
    dogs
    20
```

- If pets is a dict, then
  ```
  for k in pets:
  ```
  makes k take the values in the "keys" of the dict (the things on the left-hand-side of the : in the definition).
- `pets[k]` represents the things on the right-hand side of the ':'.

Iterables

- Something that can be in the position of x in
  ```
  for i in x:
  ```
  is called an *iterable*.
- Lists, tuples, and dicts are iterables.

Iterables have several features:

- If iter is an iterable, then list(iter) is a list consisting of all values in the iterable.
- Iterables are subject to lazy evaluation, in the sense that there are some iterables that are not actually stored in memory; they're computed.

Lazy evaluation

- For example, the strange iterable:
  ```
  range(10)
  ```
  is -- from a logical perspective -- a list [0,1,2,3,4,5,6,7,8,9].
- But in actuality, it's not that at all. It expands to that when you iterate over it.

- range(10) is something that iterates over 0-9, while list(range(10)) is a physical list that contains 0-9.
- This doesn't mean much when we're looking at range(10), but consider what it means for range(1000000000)