CSC 223 - Advanced Scientific Programming

Python Iterators

Iterators

- Python iterator syntax is used to perform repetition

  ```
  for i in range(10):
      print(i, end=' ')
  ```

- The `range` object is an *iterator*, which provides the functionality required by the `for` loop.
- A Python iterator has a built-in function `next`

  ```
  >>> I = iter([2, 4, 6, 8, 10])
  >>> print(next(I))
  2
  >>> print(next(I))
  4
  ```

Useful Iterators

- Python includes some useful iterators:
    - All the built-in data structures
    - `range`
    - `enumerate`
    - `zip`
    - `map`
    - `filter`

enumerate

- Sometimes you need to keep track of the index when iterating over a list:

```
L = [2, 4, 6, 8, 10]
for i in range(len(L)):
    print(i, L[i])
```

- The `enumerate` iterator provides this information

```
for i, val in enumerate(L):
    print(i, val)
```

zip

- The `zip` iterator allows you to iterate over multiple lists simultaneously

```
L = [2, 4, 6, 8, 10]
R = [3, 6, 9, 12, 15]
for lval, rval in zip(L, R):
    print(lval, rval)
```

- Any number of iterables can be zipped together
- The shortest length iterable determines the length of the zipped iterable.

`map`

- The `map` iterator takes a function and applies it to the values in an iterator:

```
# find the first 10 square numbers
square = lambda x: x ** 2
for val in map(square, range(10)):
    print(val, end=' ')
```

filter

- The `filter` iterator takes a boolean returning function and passes through the values for the filter function evaluates to `True`

```
# find values up to 10 that are even
is_even = lambda x: x % 2 == 0
for val in filter(is_even, range(10)):
    print(val, end=' ')
```

Iterators as Function Arguments

- Recall that `*args` and `**kwargs` can be used to pass sequences and dictionaries to functions.

```
>>> print(*range(10))
0 1 2 3 4 5 6 7 8 9
>>> print(*map(lambda x: x **2, range(10)))
0 1 4 9 16 25 36 49 64 81
```

- This means the inverse of the `zip` function is the `zip` function

```
>>> L1 = (1, 2, 3, 4)
>>> L2 = ('a', 'b', 'c', 'd')
>>> z = zip(L1, L2)
>>> print(*z)
(1, 'a') (2, 'b') (3, 'c') (4, 'd')
>>> z = zip(L1, L2)
>>> new_L1, new_L2 = zip(*z)
>>> print(new_L1, new_L2)
(1, 2, 3, 4) ('a', 'b', 'c', 'd')
```

Specialized Iterators: `itertools`

- The Python `itertools` module contains many more useful iterators
- Example:

```
>>> from itertools import combinations
>>> c = combinations(range(4), 2)
>>> print(*c)
(0, 1) (0, 2) (0, 3) (1, 2) (1, 3) (2, 3)
```