LOOPS

Python Programming — Auburn University

WHILE LOOP

- Similar to Java
- body of loop must be indented (inline body is allowed, like if)
- personally: avoid inline body unless body is a single statement.

```
x = 1
while x < 5:
    print(x)
    x += 1
```

```
x = 1
while x < 5: print(x); x += 1
```

WHILE LOOP

Preferred

- Similar to Java
- body of loop must be indented (inline body is allowed, like if)
- personally: avoid inline body unless body is a single statement.

```
x = 1
while x < 5:
    print(x)
    x += 1
```

```
x = 1
while x < 5: print(x); x += 1
```

FOR LOOP (JAVA)

- Java reminder:

```
String[] names = {“Bob”, “Sally”, “Jim”};
for(int i=0; i < names.length; i++)
   System.out.println(names[i]);
```

  is normally replaced by the foreach loop since the index is not needed:

```
String[] names = {“Bob”, “Sally”, “Jim”};
for(String name: names)
   System.out.println(name);
```

- That is, foreach loops over the contents of a collection.
- This is how the for loop works in Python…

FOR LOOP

- See example for syntax.
- Can loop over anything that is “iterable.” For now just think “any collection.”
- You can use the range function to produce sequence of integers.
  - `range(n,m)` produces sequence of integers from `n` to `m-1` in steps of `1`.
  - `range(n,m,d)` produces sequence of integers from `n` to `m-1` in steps of `d`.

```
names = ["Bob", "Sally", "Jim"]
for name in names:
    print(name)
# Output:
# Bob
# Sally
# Jim
```

```
>>> for i in range(1,10,2): print(i)
...
1
3
5
7
9
```

```
names = ["Bob", "Sally", "Jim"]
for i in range(0,len(names)):
    print(names[i])
# Output:
# Bob
# Sally
# Jim
```

ITERATING OVER SETS

- Note: items in a set are not "ordered" so, when iterating over a set the items may not appear in the order in which they were added.

```
things = {"monkey", "banana", "tree", "jungle"}
for thing in things:
    print(thing)

# Output:
# jungle
# tree
# banana
# monkey
```

ITERATING OVER DICTIONARIES

- Loops over set of **keys** in dictionary.
- The .items() method returns a collection of tuples, giving (key, value) pairs.

```
colors = {"banana": "yellow", \
          "apple": "red", \
          "lemon": "yellow", \
          "orange": "orange", \
          "grape": "purple"}

for key in colors:
    print(key, "->", colors[key])

# Output
# banana -> yellow
# apple -> red
# lemon -> yellow
# orange -> orange
# grape -> purple
```

```
for key,value in colors.items():
    print(key, "->", value)


# Same output as above
```

MODIFYING COLLECTION WHILE ITERATING

- In Java, modifying a collection while iterating over it (foreach or iterator) will result in an exception. “Fail fast iterators”
- In Python, most collections do not raise exceptions if you modify them while iterating…
- …but it is a bad idea as you may get unexpected results.

```
lst = [1, 2, 3, 4, 5]
for item in lst:
    print(item)
    if 1 in lst:
        lst.remove(1)
# Output
# 1
# 3
# 4
# 5
```