Lecture 4

Ashwini Vaidya

April 8, 2017

More loops

The `while` statement is another way of creating loops in python. Unlike `for`, the `while` statement consists of a header with a 'test' expression, which executes the program until the test is false.

```
>>> x = 'spam'
>>> while x:
... print x,
... x = x[1:]
...
spam pam am m
```

Functions

A set of statements that can be executed repeatedly in a program. Functions can output a result (but need not) and also allow us to specify various parameters as inputs to the function. Functions simply help us re-use code in an organized fashion.

Given a complex task, functions also help us break it down into manageable chunks. Each sub-problem can be executed in the form of a function. We have already used *built-in* functions e.g. `len` or `max`. Now, we will write our own functions.

```
def intersect(seq1, seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
```

```
    return res
```

If the above function is **called**, then two objects need to be **passed** into it using parentheses.

```
>>> s1 = "SPAM"
>>> s2 = "SCAM"
>>> intersect(s1, s2)
 ['S', 'A', 'M']
```

Namespace

The variable **res** is a **local** variable and is assigned within the function 'intersect'. They will be available when the function is called, but outside of that function, they don't have a meaning. Trying to print these outside the function will only result in a name error. This also means that variable names inside a function may look identical to those outside it–but their value won't be recognized outside the function–its **scope** is limited. Usually, variables outside the **def** are global variables. The place where assignment takes place, determines scope of the variable.

According to Lutz and Ascher, "When you use a name in a program, Python creates, changes, or looks up the name in what is known as a namespacea place where names live".