1 Administrative Topics

- No administrative topics today.

2 Keyword arguments

When you are defining a function, you identify parameters, and it is possible to provide a default value for that parameter. When you call that function, you can choose to provide a value for that parameter if you want to. We talked about this on Monday (Default parameter values)

```python
def sayHi(name = 'You '):
    print "Hey," , name

sayHi()
sayHi("Stephanie")
sayHi("You")
```

will print out

Hey, You
Hey, Stephanie
Hey, You

When you call a function, the value you provide as input to the function is called an argument. In the above case Stephanie is an argument. There
are two ways to provide arguments. One is using *positional* notation. This is what we have been using so far. The first argument is the value for the first parameter. The second argument is the value for the second parameter. And so on and so forth. But there is another way to do it, using *keyword* notation. In this case, you use the name of the parameter and the value of the argument together in something that looks like an assignment statement.

```python
sayHi( name = "George" )
```

For functions with many parameters, you can provide the first set of arguments by position and the last set by keyword, but once you start using keywords, you can’t stop.

In the next example, we examine code that shows the power of both default parameter values and keyword arguments. Consider the following function which has a lot of parameters, some of which have default values:

```python
# Compute the total value of entries in the column on the given number of hours on the given day.
def computeTotal(month, day, year, heading = '7M Temp', hours = 8):
    # lots of code here.
```

Then the following calls are legal:

```python
computeTotal( 5, 21, 2016, heading = '1M Temp' )
computeTotal( month=5, day=21, year=2016, hours=6) 
computeTotal( 5, 21, 2016 )
computeTotal( year = 2016, month = 5, day = 21 )
```

And the following calls are ILLEGAL:

```python
1 computeTotal( 5, 21, 2016, hours=4, '1M Temp' )
2 computeTotal( 5, 21 )
3 computeTotal( 5, 21, hours=4 )
```

Line 1 is illegal because it uses a positional argument (1) after a keyword argument.

Line 2 is illegal because it fails to provide a value for `year`. Since `year` does not have a default value, the caller must supply an argument for it. Line 3 is illegal for the same reason.
Example code

Here is an example modeled after the penguin population simulation (populations of males and females are created, and each year is simulated by making a new list). This is a much simpler simulation. All penguins survive from one year to the next. But in the bad years, there is no procreation. In the good years, every penguin has a chance of producing a new penguin. Thus, this population can grow but never shrink.

The code is written so that we never use the same name in the different functions. It is also written so that most parameters have default values. None of this is necessarily good design, but it will let us ask and answer “What if?” questions and an unambiguous way. The different names should make the scope (the scope of a variable is the region of code that can access that variable- i.e. the function that defines it).

```python
import random

def initPop(numPenguins, probF = 0.5):
    pop = []
    for i in range(numPenguins):
        if random.random() < probF:
            pop.append('f')
        else:
            pop.append('m')
    return pop

def simYear(the_pop, probBadYear = 0.2, probKids = 0.3, probFem = 0.5):
    new_pop = []
    # Shorten this!
    if random.random() < probBadYear:
        badYear = True
    else:
        badYear = False
    for penguin in the_pop:
        if badYear:
            new_pop.append(penguin)
        else:
            new_pop.append(penguin)
            if random.random() < probKids:
                if random.random() < probFem:
                    new_pop.append('f')
```

3
else:
    new_pop.append('m')
return new_pop

def runSim(numYears=30, numberOfPenguins=100, probOfKids=0.3,
            probOff=0.5, probOfBadYear=0.3):
    init_pop = initPop(numPenguins=numberOfPenguins, probF=probOff)
    for year in range(numYears):
        newer_pop = simYear(the_pop=init_pop, probFem=probOff,
                             probBadYear=probOfBadYear, probKids=probOfKids)
        init_pop = newer_pop
    return len(init_pop)

pop_size_at_end = runSim()
print pop_size_at_end

In class, we will

- Look for places where the code can be shorter
- Look at what will happen if we use default values when we shouldn’t.
- Look at what happens if we have mistakes in the code. How do we interpret messages from Python?