Arguments

Python has some nifty things you can do with the arguments to a function. Although these apply to all functions, they are particularly useful for class constructors.

First, you are allowed to provide *default values* for arguments.   If you don't supply a value for an argument, the system uses the default value.

For example, consider the function

```
def f(x, y = 2, z = 3):
        return x+y*z
```

If we call this with f(4, 5, 6) then 4 goes in for x, 5 for y and 6 for z; it returns 34.

On the other hand, if we call it with f(4, 5) then 4 goes in for x, 5 for y, and the system uses the default 3 for z; the function returns 19.

Still with

```
def f(x, y = 2, z = 3):
    return x+y*z
```

if we call f(4), then 4 goes in for x and the system uses defaults 2 for y and 3 for z; it return 10.

Finally, if we try to call f( ) then there is no value given for x and x has no default value, so we get an error message.

Once you supply one default value for an argument, all of the remaining arguments must have default values; you are not allowed to create something like

```
        def f(x, y=5, z )
```

When you call a function the arguments you supply are applied to the function parameters from left to right; any remaining parameters need to have default values.

Sometimes functions have lots of default arguments and you only want to change one. You are allowed to specify arguments in a call. For example, consider

```
def f(x, y=2, z=3 ):
    return x+y*z
```

We could call this with

```
f(5, z=4)
```

and the result will be 13: 5 goes in for x, the default 2 for y, and 4 for z.

The other nifty thing Python allows you to do with variables is to test the type of value the variable currently contains. The function that does this test is

```
isinstance(<value>, <type> )
```

The types you can use with this are

- int
- float
- str (for strings)
- list
- and any classes you create

For example we can test if the value of variable z is a string with

```
if isinstance(z, str):
        # do something with z as a string
```

When you are coding you probably know the kinds of values you have placed in variables. Where isinstance( ) is useful is in functions, where you can use it to allow the same function to be called with different kinds of variables.

For example, in Lab 09 we create a class Soundwave. The Soundwave constructor takes 4 numerical arguments that we use to compute values of a list self.samples. These arguments all have default values, so we might create a Soundwave object with

```
        Soundwave(0, 1.0, 1.0, 100)
```

or even with

```
        Soundwave( )
```

However, we sometimes want to construct a Soundwave object from an audio file, in which case we construct the self.samples list in a different way.

This means the body of the constructor looks like this
The first argument to the constructor is called "halftones". The constructor starts

```
if isinstance(halftones, str):
      self.samples = <something with file halftones>
else:
      self.samples = <something with the arguments to the constructor>
```

All of the following are ways to build Soundwaves:

```
Soundwave(0, 1.0, 1.0, 100) # makes a specific soundwave
Soundwave( )  # makes an empty soundwave
Soundwave( "mozart.wav" ) #makes a soundwave from the
                                             # file "mozart.wav"
```