Getting User Parameters

Getting User Parameters

Overview

1. Command Line Arguments: What are they?
2. sys.argv and Basic Parsing
3. arcpy GetParameter Functions
4. argparse Module
5. Other modules

Command Line Arguments: What are they?

Command Line Arguments: What are they?

- When running a python script, a user can supply arguments
    - In the call `$ python my_script.py C:\data.shp`, the program called is python, and `my_script.py C:\data.shp` parsed as the arguments to python
- We can use the fact that all input after `python` is considered an argument to allow the user to pass in arguments and options to our scripts.

sys.argv and Basic Parsing

sys.argv and Basic Parsing

- To get argument values in a python script, use `sys.argv`
  - `sys.argv` is a list of values
    - User input split on whitespace (unless argument wrapped in "")
    - In the previous example, we'd get `[r"my_script.py", r"C:\data.shp"]`
    - Script not particularly useful, so remember `sys.argv[1:]` are all arguments *to the script*

arcpy GetParameter Functions

arcpy GetParameter Functions

- Two useful functions: `arcpy.GetParameter()` and `arcpy.GetParameterAsText()`
- Both take `index` argument, and will get the argument supplied at that index
  - `GetParameter()` will return an object, assuming supplied with an object
  - `GetParameterAsText()` will return a string of the parameter
  - Unlike `sys.argv`, `index` 0 is the first script argument, not the .py file
- Easy to use when programming, but not versatile and makes terrible user interface when scripts become complicated
- Needed for creating ArcGIS toolbox tools: provides the interface between Arc and the python code

argparse Module

argparse Module

- A fully-feature std lib module for building unix-style command line interfaces
- Easily and quickly build an argument and option parser with type checking

argparse Module

Create the parser:

```
import argparse

parser = argparse.ArgumentParser(description='Description of the script.')
```

argparse Module

Add positional arguments to the parser:

```
parser.add_argument("arg_name", help="A help message.")
parser.add_argument("int_arg", type=int, help="This is an int type arg.")
```

argparse Module

Add "optional" arguments (meaning position-independent):

```
parser.add_argument("-o", "--option", type=float, default=6.325,
                    help="An optional float arg with a default of 6.325.")

parser.add_argument("-f", "--flag", action="store_true",
                    help="A bool option defualting to false.")

parser.add_argument("-r", "--required", type=str, required=True,
                    help="Required argument.")

parser.add_argument("-m", "--multiple", type=int, action="append",
                    help="Argument can be supplied multiple times;"+\
                         " produces list of all vals.")

parser.add_argument("-c" "--counter", action="count",
                    help="Counts how many times the argument was supplied.")

parser.add_arguemnt("-n", "--morethanonevalue", type=float, nargs=2,
                    help="To enter values with multiple pieces: -n 122.43 45.63")
```

argparse Module

Can define custom types using any callable that takes a string and returns desired value

```
def positive_int(int_str):
    try:
        p_int = int(int_str)
    except:
        raise argparse.ArgumentTypeError("Not an integer.")
    if p_int <= 0:
        raise argparse.ArgumentTypeError("Not a positive integer.")
    return p_int

parser.add_argument("-d", "-distance", type=positive_int)
```

argparse Module

Once the parser is setup, parse the args:

```
args = parser.parse_args(sys.argv[1:])  # or any list

# returns an argparse namespace object
# interface is like a class
# args.distance, args.option, etc.

# can also get args in a dictionary:
args_dict = vars(args)
```

Other Modules

Other Modules

- Std Lib:
  - argparse
  - optparse
  - getopt
- click (my favorite)
- docopt (haven't used, but looks interesting)
- Certainly many more…