Scopes & Namespaces continued

Local namespace: the current executing function
Global namespace: the module in which this function was defined
Named namespaces: Referenced from global or local namespace

Recall ...

import grid
def fill(cave, row, cell):
    ... grid.fill_cell(row, col, grid.white)
Also recall ... referring to global (module) variables

```python
def some_function( ):  
global x  
x = y  
Search for y  
return x + y  
Create a local x – Search for a global x in this module
```

"global" means: I want to use and change the variable in module scope, not create a new one.

---

Aside ... exceptions

What if the user gives a file name that doesn’t exist?

What if the user types letters when a number is expected?

We could check for everything, everywhere, but our code would get awfully messy.

Exceptions help us organize the extra code and make it easier to read.

---

""

game_dict: ...
""

def read( filename, min_length=3 ):  
  """ ... function interface defined here  
  """
  global dict  
dict = [ ]  
...  

def search( str ):  
  """ ... function interface specified here  
  """
  # FIXME: Search the dictionary

dict is global (in the game_dict module) so that we can read it once and use it each time we call search( )

---

Rules of thumb:

Use exceptions only for bad things, never for normal control flow

Use exceptions to separate standard processing from “oops” processing

Their purpose is to make code cleaner, easier to read and to get right.

Never handle exceptions silently (terrible for debugging)
Coming next week ... classes

Namespaces are great. I want more! More!

(And I don’t want to write a different module file for each one. I want to have a bunch of objects, each with their own namespace.)

A class is a lot like a module.
From one class, we make as many objects as we want. Each one has its own namespace (“attributes”, including functions and variables)

Summary: It’s all about controlling complexity

Scope and local variables help us treat functions as black boxes: Read the spec (docstring), not the code.
Modules helps us group functions (often with shared data) into chunks we can manage separately.
Exceptions help us separate normal flow from “what if” flow, and keep it from tangling our code.