AMS 595: Fundamentals of Computing: Part II

Lecture 3: Control Flow and Functions

Xiangmin Jiao

Stony Brook University

Control Flow

- Conditionals: `if-elif-else` branches
    - Nonzero value and nonempty containers are treated as `True`
    - Zero value and empty containers are treated as `False`
    - Special value `None` is treated as `False`
    - No switch/case statement in Python
- The `while`-loop is similar to MATLAB
- The `for`-loop typically loops through a range, a list, or a dictionary
- The `break` and `continue` statements are similar to MATLAB
- List comprehensions provide a compact way to initialize lists
- Demo: Jupyter notebook on control flows

Functions

- Defined by

```
def fun_name(arglist):
    '''doc-string'''
    statements ...
```

- Arguments
  - Mutable objects are passed by reference
  - Can have optional arguments with default values
  - Mutable default arguments are evaluated only once; immutable default arguments are evaluated for each call
- Return values
  - Can return an object (reference)
  - Can return multiple values via a tuple
  - If no explicit return value, then return `None`
- A function has its doc-string, used by the help system
- Demo: Jupyter notebook on functions

Scope and Namespace

- A *namespace* is a mapping from names to objects. A *scope* is a region of a Python program where a namespace is directly accessible.
- Each module (or file) has its own “global” namespace
- Each function has their own namespaces
- Within a function, if a variable is not found local scope, Python searches the variable in its enclosing scope (e.g., the global namespace of the module containing the function)
- Python allows nested functions, like MATLAB
  - Nested function is only accessible from parent function
  - However, unlike MATLAB, nested sub-function cannot access variables in parent function
- To learn more about scope, see `Python documentation`

Recursion vs. Iteration

- Recursion (via recursive functions) and iteration (via loops) can both achieve repetition in programming
- Function calls involve creating and maintaining stack frames (see `visualization`)
  - Recursion is more expensive and uses more memory than iteration
  - Depth of recursion is limited by available memory
- Recursion can be converted to iteration for faster execution
  - Approach 1: Brute-force using stack to simulate stack frames
  - Approach 2: Manual conversion to *tail calls*
    1. Key: Convert recursive calls to *tail calls* (function call as last statement)
    2. Enclose function body in "`while True:`"
    3. Replace recursive tail call "`f(x=x1, y=y1, ...)`" with "`x, y, ... = x1, y1, ...`"
    4. Clean up the code
- Demo: Jupyter notebook on recursive functions

Lambda Functions

- Lambda functions as disposable, nameless functions
- For example:

  ```
  g = lambda x: x**2
  ```

  which is equivalent to

  ```
  def g (x):
      return x**2
  ```

- Lambda functions are often used as arguments to other functions (such as `sort` and `filter`)
- Lambda functions also have their own call frames
- Demo: Jupyter notebook on Lambda functions

Style Guide for Python Code

Most projects use PEP 8 style; most editors enforce it automatically

- Use **4-space indentation**, and no tabs.
- **Wrap lines** so that they don't exceed 79 characters; long lines can be broken with the \ character
- Use **blank lines** to separate functions and classes, and larger blocks of code inside functions.
- When possible, put **comments** on a line of their own.
- Use **docstrings** for the help system.
- Use **spaces** around operators and after commas, but not directly inside bracketing constructs: $a = f(1, 2) + g(3, 4)$.
- Use **lower_case_with_underscores** for functions and method; (use **CamelCase** for classes)
- Don't use fancy encodings unless you have to; don't use non-ASCII characters in identifiers