Procedural Programming

Lect. PhD. Arthur Molnar

Babes-Bolyai University

arthur@cs.ubbcluj.ro

Overview

Procedural programming

- A **programming paradigm** is a fundamental style of computer programming.
- **Imperative programming** is a programming paradigm that describes computation in terms of statements that change a program state.
- **Procedural programming** is imperative programming in which the program is built from one or more procedures (also known as subroutines or functions).

What is a function

A self contained block of statements that:

- Has a *name*,
- May have a list of (formal) *parameters*,
- May *return* a value
- Has a specification which consists of:
  - A *short description*
  - *Type and description of parameters*
  - Conditions imposed over input parameters (*precondition*)
  - Type and description for the return value
  - Conditions that must be true after execution (*post-condition*).
  - Any Exceptions raised

What is a function


```
def maximum(x,y):
    """
    Return the maximum of two values
    input: x,y - the parameters to compare
    output: The largest of the parameters
    Error: TypeError - parameters cannot be compared
    """
    if x > y:
        return x
    return y
```

What is a function

- Can you tell what the function below does?
- Did it take more than a few seconds?

```
def f(c):
    b = []
    while not sol(b) and c != []:
        cand = next(c)
        c.remove(cand)
        if acceptable(b + [cand]):
            b.append(cand)
    if sol(b):
        found(b)
    return None
```

NB!

A function without specification is not complete!

What is a function

Every non-UI function written by you should:

- Use meaningful names (function name, variable names)
- Provide specification
- Include comments
- Have a test function (will come later)

What is a function


```
def greedy(c):
    '''
    Generic greedy algorithm
    input: c - set of candidates
    output: solution of generic problem
    '''
    # The empty set is the candidate solution
    b = []
    while not solution(b) and c != []:
        # Select best candidate (local optimum)
        candidate = selectMostPromising(c)
        c.remove(candidate)
        # If the candidate is acceptable, add it
        if acceptable(b + [candidate]):
            b.append(candidate)
    if solution(b):
        return b
    # In case no solution
    return None
```

What is a function

- A **function definition** is an executable statement introduced using the keyword **def**.
- The function definition does not execute the function body; this gets executed only when the function is called. A function definition defines a user-defined function object.

```
def maximum(x,y):
    """
    Return the maximum of two values
    input: x,y - the parameters to compare
    output: The largest of the parameters
    Error: TypeError - parameters cannot be compared
    """
    if x > y:
        return x
    return y
```

Variable scope

A *scope* defines the visibility of a name within a block. If a local variable is defined in a block, its scope includes that block. All variables defined at a particular indentation level or scope are considered local to that indentation level or scope

- Local variable
- Global variable

Demo

Variable scope

ex04_VariableScope.py

Variable scope

Rules to determine the scope of a particular name (variable, function name):

- A name defined inside a block is visible only inside that block
- Formal parameters belong to the scope of the function body (visible only inside the function)
- A name defined outside a function (at the module level) belongs to the module scope
- When a name is used in a code block, it is resolved using the nearest enclosing scope.

Variable scope

At any time during execution, names are resolved using:

- The innermost scope, which is searched first, contains the local names (inside the block)
- The scopes of any enclosing functions, which are searched starting with the nearest enclosing scope
- The next-to-last scope contains the current module's global names
- The outermost scope (searched last) is the namespace containing built-in names

Variable scope

- Use the **globals()** and **locals()** functions to figure out the scope of each variable

Recap

What other python built-in functions do you know?

Calls

A **block** is a piece of Python program text that is executed as a unit. Blocks of code are denoted by line indentation. A **function body** is a block. A block is executed in an *execution frame*. When a function is invoked a new execution frame is created.

Execution frames

`http://www.pythontutor.com/visualize.html`

Calls

An execution frame contains:

- Some administrative information (used for debugging)
- Determines where and how execution continues after the code block's execution has completed
- Defines two namespaces, the local and the global namespace, that affect execution of the code block.
- A *namespace* is a mapping from names (identifiers) to objects. A particular namespace may be referenced by more than one execution frame, and from other places as well.

Calls

- Adding a name to a namespace is called binding a name (to an object); changing the mapping of a name is called rebinding.
- Removing a name is unbinding.
- Namespaces are functionally equivalent to dictionaries (and often implemented as dictionaries).

Discussion

What did the output of locals(), globals() look like?

Parameter passing

- **Formal parameter** - an identifier for an input parameter of a function. Each call to the function must supply a corresponding value (argument) for each mandatory parameter
- **Actual parameter** - a value provided by the caller of the function for a formal parameter.
- The actual parameters (arguments) to a function call are introduced in the local symbol table of the called function when it is called (arguments are passed *by object reference*)

Parameter passing

- **Pass by value** - the argument is evaluated, and a copy of the evaluation result is bound to the formal parameter of the function
- **Pass by reference** - function receives a reference to the actual argument, rather than a copy to its value
- **Side effect** - a function that modifies the caller's environment (beside producing a value) is said to have side effects

Demo

Parameter passing

ex05_ParameterPassing.py

Parameter passing

Discussion

What are the advantages and disadvantages of pass by value and pass by reference?

Parameter passing

How about in Python?

Object references are passed by value

Passing parameters

What happened in the studied example?

- At first, Python behaves like call-by-reference
- When you change a variable's value, it "switches" to call-by-value

Demo

Side Effects

ex06_SideEffects.py

A Working Program

ex07_RationalCalculator.py