§4.1-4.3: Procedures

- *devo*

22 Sep 2006
CMPT14x
Dr. Sean Ho
Trinity Western University

Review last time (§3.4-3.10, 5.4)

- String concatenation (+), repetition (*)
- Qualified import
- while loops: continue, break, else
- Common mistakes in loops
- for loops
- range()

What's on for today (§4.1-4.3)

- Procedures (functions, subroutines)
  - No parameters
  - With parameters
  - Scope
  - Global variables (why not to use them)
  - Call-by-value vs call-by-reference

Procedures

- Fourth program structure/flow abstraction is composition
- This is implemented in Python using procedures
  - Also called functions, subroutines
- A procedure is a chunk of code doing a sub-task
  - Written once, can be used many times
- We've already been using procedures:
  - print, input, raw_input, etc. (not if or while)

Procedure input and output

- Procedures can do the same thing every time:
    - print # prints a new line
- Or they can change behaviour depending on parameters (arguments) input to the procedure:
    - print(“Hello!”) # prints the string parameter
  - List of parameters goes in parentheses
    - (print is special and doesn't always need parens)
- Procedures can also return a value for use in an expression:
    - numApples = input(“How many apples? ”)

Example: no parameters

- Procedure to print program usage info:

```
def print_usage():
    ”””Display a short help text to the user.”””
    print “This program calculates the volume”,
    print “of a sphere, given its radius.”
```

docstring

```
...
if string.capitalize(userInput) == “H”:
    print_usage()
```

Example: with parameters

- Calculate volume of a sphere:

```
from math import pi
def print_sphere_volume(radius):
    ”””Calculate and print the volume of a sphere
    given its radius.
    ”””
    print “Sphere Volume = %.2f” % (4/3)*pi*(radius**3)

print_sphere_volume(3.5)
```

formal parameter

actual parameter

Scope

- Procedures inherit declarations from enclosing procedures/modules:
  - Declarations:
    - import (e.g., math.pi)
    - variables
    - Other procedures

- Items declared within the procedure are local: not visible outside that procedure
- The scope of a variable is where that variable is visible

Example: scope

```
from math import pi
def print_sphere_volume(radius):
    ”””Calculate and print the volume of a sphere
    given its radius.
    ”””
    vol = (4/3)*pi*(radius**3)
    print “Sphere Volume = %.2f” % vol

myRadius = 3.5
print_sphere_volume(myRadius)
```

radius, vol, pi, myRadius

myRadius, pi

- What variables are visible in print_sphere_volume()?
- What variables are visible outside the procedure?

Keep global variables to a minimum

```
from math import pi
def print_sphere_volume(radius):
    ”””Calculate and print the volume of a sphere
    given its radius.
    ”””
    myVolume = (4/3)*pi*(radius**3)
    print “Sphere Volume = %.2f” % myVolume

myVolume = 10
print_sphere_volume(3.5)
```

Note assignment to global var

What is the value of myVolume?

Call-by-value and call-by-reference

- In other languages procedures can have side effects: (M2)

```
PROCEDURE DoubleThis(VAR x: INT);
BEGIN
    x := x * 2;
END DoubleThis;

numApples := 5;
DoubleThis(numApples);
```

- Call-by-value means that the value in the actual parameter is copied into the formal parameter
- Call-by-reference means that the formal parameter is a reference to the actual parameter, so it can modify the value of the actual parameter (side effects)

Python is both CBV and CBR

- In M2, parameters are call-by-value
  - Unless the formal parameter is prefixed with “VAR”: then it's call-by-reference
- In C, parameters are call-by-value
  - But you can make a parameter be a “pointer”
- Python is a little complicated: roughly speaking,
  - Immutable objects (7, -3.5, False) are call-by-value
  - Mutable objects (lists, user-defined objects) are call-by-reference

Example of CBV in Python

```
def double_this(x):
    ”””Double whatever is passed as a parameter.”””
    x *= 2

numApples = 5
double_this(5)                         # x == 10
double_this(numApples)                 # x == 10
double_this(“Hello”)                   # x == “HelloHello”
```

- double_this() has the ability to modify the global numApples, but it doesn't because the changes are only done to the local formal parameter x.

Summary of today (§4.1-4.3)

- Procedures (functions, subroutines)
  - No parameters
  - With parameters
  - Scope
  - Global variables (why not to use them)
  - Call-by-value vs call-by-reference

TODO

- Quiz ch3 on Mon
- Lab02 due next MTW: 3.14 # 36 and 45
- Read through §4.7 and Py ch5 for Mon