§4.8-4.10, Py ch5-6: Recursion

6 October 2008
CMPT14x
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Review from last time (ch4)

- Some debugging tips
- A fun example: ROT13
  - ord(), chr(), string indexing, len()
  - Stub program

Addendum: iterating a string

- Iterating through a string:

```
for idx in range(len(myString)):
    myChar = myString[idx]
```

  - Shorthand in Python:
    (can treat strings as lists of characters)

```
for myChar in myString:
    myChar ...
```

  - For example:

```
for myChar in "Hello World!":
    print myChar
```

What's on for today (§4.8, Py ch5-6)

- Recursive functions
  - Factorial example
- Call stack, backtrace
  - Fibonacci example
- Abstract Data Types
  - Type hierarchy
- Enumerations

Recursion

- Recursion is when a function invokes itself
- Classic example: factorial (!)
  - n! = n(n–1)(n–2)(n–3) … (3)(2)(1)
  - 0! = 1
- Compute recursively:
  - Inductive step: n! = n*(n–1)!
  - Base case: 0! = 1
- Inductive step: assume (n–1)! is calculated correctly; then we can find n!
- Base case is needed to tell us where to start

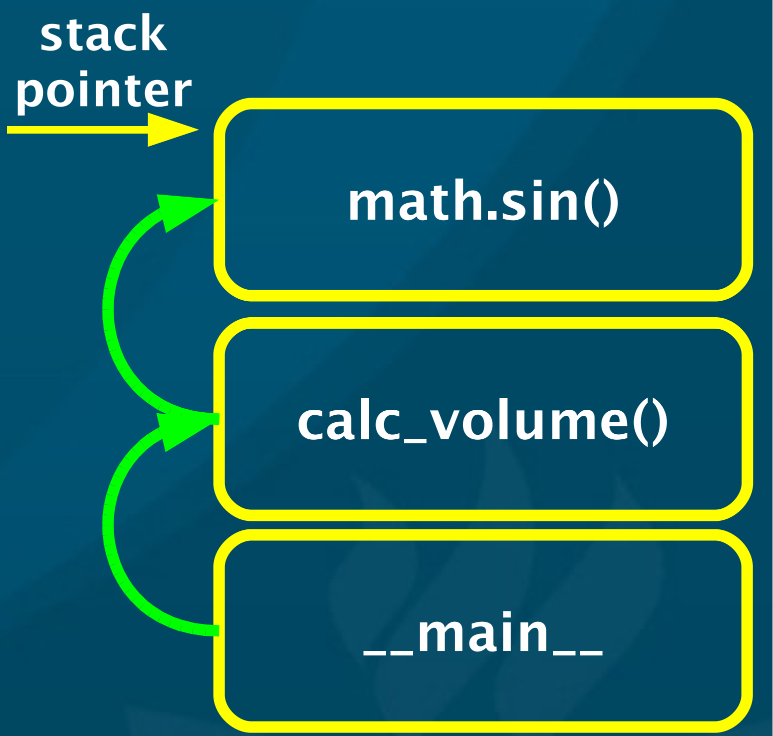
factorial() in Python

```
def factorial(n):
    """Calculate n!.  n should be a positive integer."""
    if n == 0:                          # base case
        return 1
    else:                               # inductive step
        return n * factorial(n-1)
```

- Progress is made each time: factorial(n-1)
- Base case prevents infinite recursion
- What about factorial(-1)? Or factorial(2.5)?

The call stack

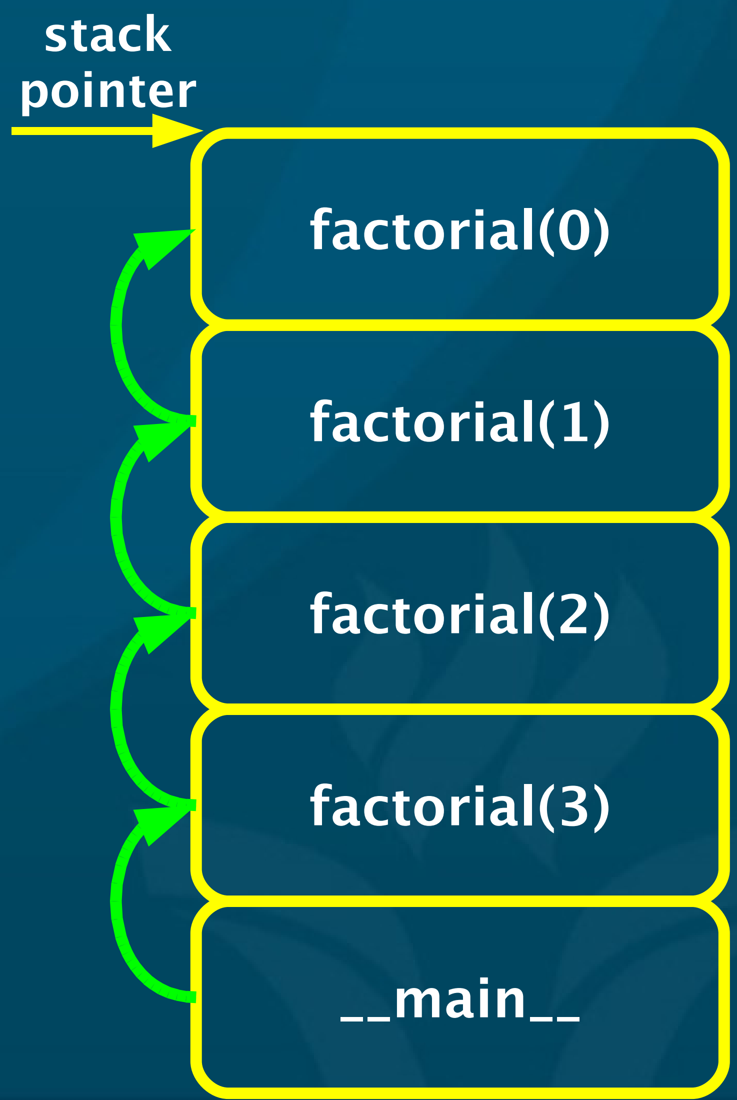
- When a program is running, an area of memory is set aside to store local variables, the state of the program, etc.
- When a procedure is invoked, the calling context is saved, and a new chunk of memory is allocated for the procedure to use: its stack frame
- When the procedure finishes, its frame is released and control goes back to the calling context
- The stack pointer keeps track of what frame is currently running



Call stack for recursive functions

```
def factorial(n):
    """Compute the factorial of a
      positive integer."""
    if n == 0:
        return 1
    else:
        return n*factorial(n-1)
```

- If there were any local variables, each frame would have its own instance of the local variables
- When an error (exception) happens, IDLE shows a backtrace: part of the call stack



Another recursive ex.: Fibonacci

- Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34,...
  - Each number is the sum of the two previous

```
def fibonacci(n):
    """Compute the n-th Fibonnaci number.
    pre: n should be a positive integer.
    """
    if n == 0 or n == 1:                # base case
        return 1
    else:                               # inductive step
        return fibonacci(n-2) + fibonacci(n-1)
```

  - Note: very inefficient algorithm!

Abstract Data Types

- Recall the categorization of
  - Atomic vs. Aggregate (compound) types
- Some examples of atomic data types:
  - Real (float), integer (int), Boolean (bool)
  - Character (if the language has such a type)
- Some examples of aggregate data types:
  - Arrays, tuples, dictionaries, records/structs
- Abstract Data Type (ADT):
  - Details of implementation are hidden from user (how to represent a float in binary form?)

M2 type hierarchy (partial)

- Atomic types
  - Scalar types
    - Real types (REAL, LONGREAL)
    - Ordinal types (CHAR)
      - Whole number types (INTEGER, CARDINAL)
      - Enumerations (§5.2.1) (BOOLEAN)
      - Subranges (§5.2.2)
- Structured (aggregate) types
  - Arrays (§5.3)
    - Strings (§5.3.1)
  - Sets (§9.2-9.6)
  - Records (§9.7-9.12)
- Also can have user-defined types

Python type hierarchy (partial)

- Atomic types
  - Numbers
    - Integers (int, long, bool): 5, 500000L, True
    - Reals (float) (only double-precision): 5.0
    - Complex numbers (complex): 5+2j
- Container (aggregate) types
  - Immutable sequences
    - Strings (str): "Hello"
    - Tuples (tuple): (2, 5.0, "hi")
  - Mutable sequences
    - Lists (list): [2, 5.0, "hi"]
  - Mappings
    - Dictionaries (dict): {"apple": 5, "orange": 8}

Enumeration types in M2 (also C)

```
TYPE
    DayName = (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
VAR
    today : DayName;
BEGIN
    today := Mon;
```

- We could have used CARDINALs instead (and indeed the underlying implementation does)
  - But the logical semantic of today's type is a DayName type, not a CARDINAL
- Can be thought of as Sun=0, Mon=1, Tue=2, ...
- No explicit enumeration scheme in Python

Review of today (§4.8, Py ch5-6)

- Recursive functions
  - Factorial example
- Call stack, backtrace
  - Fibonacci example
- Abstract Data Types
  - Type hierarchy
- Enumerations