Scripting & Computer Environments

Core Python: Functions

IIIT-H

Function

A named code block with well-defined role.

- So far, some built-in functions:

  `len(), abs(), int(), append(), etc`

- Why functions?
  - Maximize code reuse
  - Minimize code redundancy
  - Code readability
  - Easy debugging

Defining Functions

```
def <name>(parameter list):          # optional list
     <DocString>                     # documentation string
     <statements>
     return <value>                  # optional
```

- `def` statement creates an object and assigns it to `<name>` (much like '=').
- Function exists only after `def` has been executed at *runtime*.
- `Docstring` (optional) provides convenient way of associating documentation with the function `<name>`.

Defining Functions (2)

- Gives a name, specifies parameteres & structures the blocks.

Examples

```
>>>def hello():
      "prints hello message"
      print "Hello World!"


>>>def add(x,y):
      "Adds two objects"
      return x+y
```

```
>>>print hello.__doc__

>>>print add.__doc__

>>>help(hello)

>>>help(add)
```

Function Calling

```
>>>add(10,20)

>>>add(3)
                                        # polymorphism in action
>>>add('Hi', 'Bye')

>>>L=add([1,2,3], [4,5,6])
```

Examples

1. WAF named `common()` to find the intersection of two sequence types (e.g. lists, tuples).

2. WAF named `Fib()` to print the first N elements of a Fibonacci series.

Functions: Namspaces

- Namespace/Context: a place where names live (e.g. function names and identifiers).
- Is a naming system for making names unique to avoid collision.

  e.g. directory structure of file systems, nodes in a network
- Same identifier can be defined in multiple namespaces.
- The place where you assign a name in your code determines the namespace it will be in.

- Python namespaces:

  1. Global names - of a module
  2. Local names - in functions/methods
  3. Built-in names - built-in functions and exceptions

- Scope: the area of a program where a name can be unambiguously used (such as inside functions)
- visibility of a variable.
- Python's name resolution uses *the LEGB lookup rule*:

  Local (L) → Enclosing functions if any (E) → Global (G) → then built-in (B).
- Local vs Global scopes

Examples

```
>>>S='I am global'
>>>def f():
       print S
>>>f()                                    # calling f()...
```

Examples

```
>>>S='I am global'
>>>def f():
       S='I am Local'
       print S
>>>f()                                    # calling f()...
>>>print S
```

Examples

```
>>>S='I am global'
>>>def f():
      print S
      S='I am now local'
      print S
>>>f()                                     # output??
```

Passing Arguments

Arguments

- Simply, inputs to functions.
- Are references to objects sent by the caller function (Python).
- Pass-by-assignment/pass-by-object-reference (Python)

- For *immutable arguments* (e.g. integers, strings, tuples), the passing acts like pass-by-value.
- For *mutable arguments* (e.g. lists, dictionaries), it acts like pass-by-reference.
- Command-line arguments are in the list `sys.argv`. (Read about the `getopt` module).

Argument-Matching Modes

1. Required (Positional) Arguments

   Syntax: `func(value)`

   - Matching is by position.
   - # of args in function definition should match with the caller's.

2. Keyword Arguments

   Syntax: `func(name=value)`

   - Matching is by name (keyword).
   - Order does not matter.
   - Caller identifies arguments by the parameter name.

Argument-Matching Modes (2)

3. Default Arguments

   Syntax: `def func(name=default_value):`

   - Assumes a default value if no value is provided in the call.

4. Variable-length Arguments

   Syntax: `def func(some_args, *var_args_tuple):`

   - All the arguments need not be specified during definition.
   - When called with more arguments, the non-specified (variable) arguments are collected in the `var_args_tuple` variable.

Recursive Functions

Recursion (Latin: Recurrō)

- To run back or return to self.
- Recursive functions call themselves, directly or indirectly.

- Recursion in natural languages

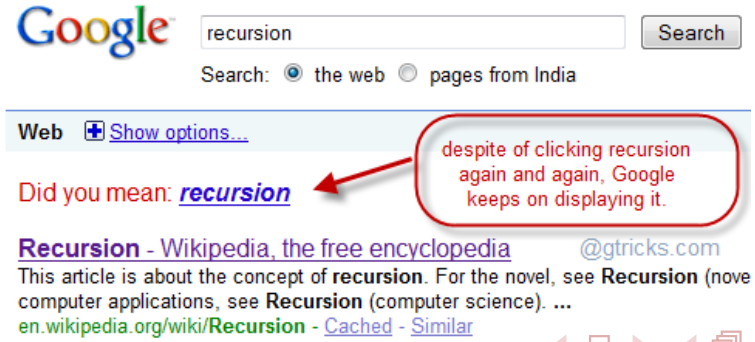
I know the answer.

He thinks that I know the answer.

She thinks that he thinks that I know the answer.

They think that she thinks that he thinks that I know the answer. etc...

- Recursion according to Google :)



Factorial

$$Fact(n) = \begin{cases} 1 & \text{if } ?? \\ ?? & \text{Otherwise} \end{cases}$$

Sum of the first n natural Numbers

$$Sum(n) = \begin{cases} 0 & \text{if } ?? \\ ?? & \text{Otherwise} \end{cases}$$

Recursive String Reversal

$$Reverse(str) = \begin{cases} ?? & \text{if empty string} \\ ?? & \text{Otherwise} \end{cases}$$

Factorial

$$Fact(n) = \begin{cases} 1 & \text{if ??} \\ ?? & \text{Otherwise} \end{cases}$$

Sum of the first n natural Numbers

$$Sum(n) = \begin{cases} 0 & \text{if ??} \\ ?? & \text{Otherwise} \end{cases}$$

Recursive String Reversal

$$Reverse(str) = \begin{cases} ?? & \text{if empty string} \\ ?? & \text{Otherwise} \end{cases}$$

Factorial

$$Fact(n) = \begin{cases} 1 & \text{if ??} \\ ?? & \text{Otherwise} \end{cases}$$

Sum of the first n natural Numbers

$$Sum(n) = \begin{cases} 0 & \text{if ??} \\ ?? & \text{Otherwise} \end{cases}$$

Recursive String Reversal

$$Reverse(str) = \begin{cases} ?? & \text{if empty string} \\ ?? & \text{Otherwise} \end{cases}$$

`a` power `n`

$$a^n = \begin{cases} 1 & \text{if n=0} \\ ?? & \text{if n is even} \\ ?? & \text{if n is odd} \end{cases}$$

Combinatorics: n choose k

$$C(n,k) = \begin{cases} 1 & \text{if k=0 or n=k} \\ C(n-1,k) + C(n-1,k-1) & \text{Otherwise} \end{cases}$$

`a` power `n`

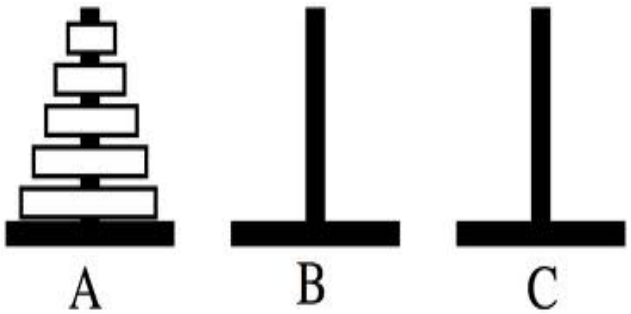
$$a^n = \begin{cases} 1 & \text{if n=0} \\ ?? & \text{if n is even} \\ ?? & \text{if n is odd} \end{cases}$$

Combinatorics: n choose k

$$C(n,k) = \begin{cases} 1 & \text{if k=0 or n=k} \\ C(n-1,k) + C(n-1,k-1) & \text{Otherwise} \end{cases}$$

Tower of Hanoi

Goal: To transfer n disks from A to C using B as a temporary location.



Rules:

- Move only one disk at a time.
- Never put a larger disk on top of a smaller.

Generally, # of moves required for n disks= 2^n-1

Modules

http://docs.python.org/2/tutorial/modules.html