Introduction to Programming Language (ITP101)

Functions

Mulualem Teku

GCIT, Bhutan

Sep 2019

...Previously & Today...

- Intro to Python ✓
  - `Data types` (e.g. Numbers)
  - `Operators`
  - `Variables`
  - `Expressions`
  - `Statements`
  - `Input/output`
- Control structures ✓
  - `if...else`, `if...elif...else`
  - `for loop`
  - `while loop`

Today:

- Functions

Brainstorm

1. Output? (assume Python 3)

```
5 // 2 * 10 / 3
```

```
3 ** 2 + 6 * 1 - 4
```

```
n = 2 < 5 and 3!=3

if n:
   print "Eureka!"
else:
   print "Yalama!"
```

```
for i in range(50):
    print i+3
```

```
for i in range(1, 51):
    if i % 5 == 0:
         continue
    elif i == 40:
         break
    else:
         print i
print "Kuzu"
```

```
x = 4
y = 5
while x < 10:
    y = y + x
    x = x + 2
print y
```

Recap: Data Types

- Everything in Python is an object.

Some Python data types (objects)

- Numbers
- Functions ✓
- Lists
- Tuples
- Dictionaries
- Strings
- Files
- Modules
- Classes

- Mutable vs Immutable objects

Functions: The Whats & Whys

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, etc

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 parameters & structures the blocks.

Example

```
>>>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)
```

Returning multiple values?

Function Calling

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

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

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

Examples

Examples

1.
2.
3.
4.
5.

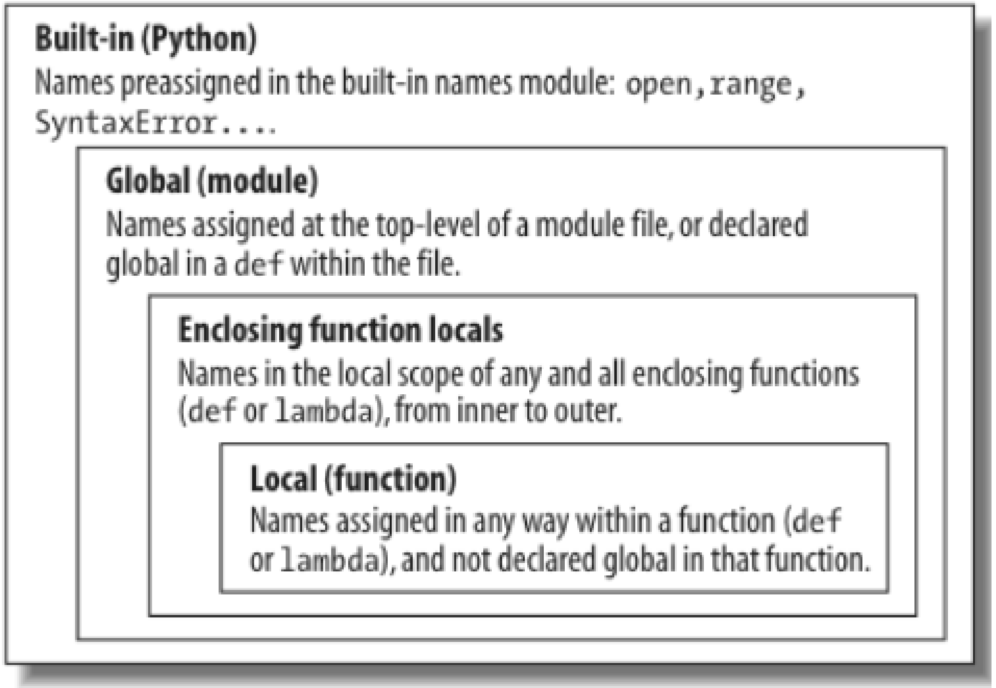
Functions: Scopes

- Scope: the area of a program where a name can be unambiguously used (such as inside functions).

- Is visibility of a variable.

- Python's name resolution uses *the LEGB lookup rule*:

1. Local (L)
2. Enclosing functions if any (E)
3. Global (G)
4. Built-in (B)



Built-in (Python)
Names preassigned in the built-in names module: `open`, `range`, `SyntaxError`....

Global (module)
Names assigned at the top-level of a module file, or declared global in a `def` within the file.

Enclosing function locals
Names in the local scope of any and all enclosing functions (`def` or `lambda`), from inner to outer.

Local (function)
Names assigned in any way within a function (`def` or `lambda`), and not declared global in that function.

Local vs Global Scopes

Example

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

Example

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

Example

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

Local vs Global Scopes

Example

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

Example

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

Example

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

Local vs Global Scopes

Example

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

Example

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

Example

```
>>>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

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.

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

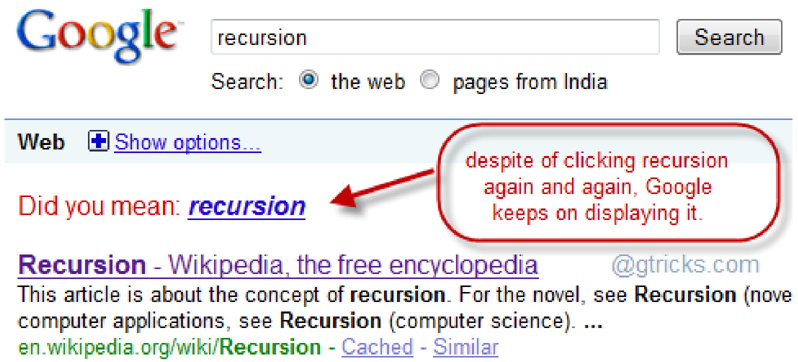
Recursion (Latin: Recurrō)

- 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 - Google's way :)



recursion Search

Search: ● the web ○ pages from India

Web ⊞ Show options...

Did you mean: ***recursion***

despite of clicking recursion again and again, Google keeps on displaying it.

Recursion - Wikipedia, the free encyclopedia @gtricks.com

This article is about the concept of **recursion**. For the novel, see **Recursion** (nove

computer applications, see **Recursion** (computer science). **...**

en.wikipedia.org/wiki/**Recursion** - Cached - Similar

Examples

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

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

Examples

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

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

Examples

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`

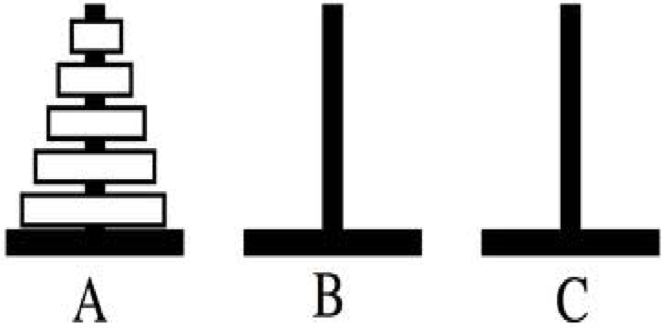
$$a^n = \begin{cases} 1 & \text{if n=0} \\ ?? & \text{if n is even} \\ ?? & \text{if n is odd} \end{cases}$$

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