

§10.0-10.7, Py tut §9.0-9.2: Namespaces and Scope

14 Nov 2008

CMPT14x

Dr. Sean Ho

Trinity Western University

Quiz07

- Contrast **procedural** vs. **object-oriented** programming.
- Define: **object**, **class**, **instance**, **attribute**, **method**
- Come up with an **application** for records: describe in English what the record represents, what information it holds, and what it would be used for (what operations you might perform on such a record).
Define (in pseudocode) a suitable **record type**, including the **type** of each record field.

Namespaces

- A **namespace** is a mapping from **names** (identifiers) to **objects**
 - **math.pi** is a **mapping** from the name '**pi**' to the float object **3.1415926535...**
 - **math.pi** is in the **namespace** provided by the **math** standard library module
- At a given point in the execution of a program, any number of namespaces may be **current**:
 - Defines what names are **valid** at that point

Creating namespaces

- The **default** namespace is present as long as the Python interpreter/compiler is active
 - Contains **built-in** names like **abs()**, **float()**, **ZeroDivisionError**, etc.
- Each **module** has a **global** namespace visible everywhere in that module
 - Variables defined in the outermost level of your Python **file**
- Each **function** invocation and **class** definition also defines a new **local** namespace
 - Can be **nested**

Namespaces avoid name collision

- The point of namespaces is to avoid **name collision**:
- Names defined in one namespace do not **conflict** with names defined in another namespace

```
import math
```

```
print math.pi          # namespace of math module
```

```
pi = 3                 # namespace of current file:
```

```
    __main__
```

- Two **libraries**, or two **classes**, can define functions with the **same** name without conflict

- `complex.add()` and `Fraction.add()`

Example of namespaces

```
G1 = 'global'
```

```
def factorial(n):  
    L1 = 'local'  
    if n == 0 or n == 1:  
        return 1  
    return n * factorial(n-  
1)
```

File module's global namespace (`__main__`)

Local namespace for each call to factorial

Scope

- “A **scope** is a **textual** region of a Python program where a namespace is **directly accessible**.”
 - Can access without using **module** name
 - ◆ e.g., **pi** rather than **math.pi**
- Scope deals with the **order** in which namespaces are searched to **resolve** a name
 - First search **local** scope
 - Then search **enclosing** functions/classes
 - Then search **global** scope for that file/module
 - Then search **built-in** names

New names add to local scope

- New names are **created** by:
 - **Assignment**: `x = 5`
 - **Function definitions**: `def factorial(n):`
 - **Class definitions**: `class Fraction:`
 - **Imports**: `from math import *`
- **New** names always add to the **local** scope

```
def distance(x1, y1, x2, y2):  
    from math import sqrt  
    return sqrt((x2-x1)**2 + (y2-y1)**2)  
sqrt                # not defined here!
```


The *global* directive

- Names outside the **local** scope are **read-only**
 - Attempts to **modify** them result in creating a new **local copy**

```
G1 = 'global'  
def fun():  
    G1 = 'local'      # creates local copy of G1  
fun()  
G1                  # G1 is unchanged
```

- The **global** directive says that references to those names refer to the file/module's **global** scope