Introduction to Python - Continued

ELTE - Budapest- September 2022

Classes

class *name*:

"*documentation*"

statements

-or-

class *name*(*base1*, *base2*, ...):

...

Most, *statements* are method definitions:

def *name*(self, *arg1*, *arg2*, ...):

...

May also be *class variable* assignments

Example Class

```
class Stack:
    "A well-known data structure…"
    def __init__(self):             # constructor
        self.items = []
    def push(self, x):
        self.items.append(x)        # the sky is the limit
    def pop(self):
        x = self.items[-1]          # what happens if it's empty?
        del self.items[-1]
        return x
    def empty(self):
        return len(self.items) == 0     # Boolean result
```

Using Classes

- To create an instance, simply call the class object:

```
x = Stack()    # no 'new' operator!
```

- To use methods of the instance, call using dot notation:

```
x.empty()    # -> 1
x.push(1)                    # [1]
x.empty()    # -> 0
x.push("hello")                    # [1, "hello"]
x.pop()              # -> "hello"  # [1]
```

- To inspect instance variables, use dot notation:

```
x.items              # -> [1]
```

Subclassing

```
class FancyStack(Stack):
   "stack with added ability to inspect inferior stack items"

   def peek(self, n):
      "peek(0) returns top; peek(-1) returns item below that; etc."
      size = len(self.items)
      assert 0 <= n < size                    # test precondition
      return self.items[size-1-n]
```

Subclassing (2)

```
class LimitedStack(FancyStack):
    "fancy stack with limit on stack size"

    def __init__(self, limit):
        self.limit = limit
        FancyStack.__init__(self)        # base class
      constructor

    def push(self, x):
        assert len(self.items) < self.limit
        FancyStack.push(self, x)         # "super" method call
```

Class / Instance Variables

```
class Connection:
    verbose = 0                    # class variable
    def __init__(self, host):
        self.host = host           # instance variable
    def debug(self, v):
        self.verbose = v           # make instance variable!
    def connect(self):
        if self.verbose:           # class or instance variable?
            print "connecting to", self.host
```

Instance Variable Rules

- On use via instance (self.x), search order:
  - (1) instance, (2) class, (3) base classes
  - this also works for method lookup
- On assignment via instance (self.x = ...):
  - always makes an instance variable
- Class variables "default" for instance variables
- But...!
  - mutable *class* variable: one copy *shared* by all
  - mutable *instance* variable: each instance its own