3-ClassesAndObjects

December 10, 2014

```
In [1]: def adder(a, b):
            x = 10
            result = a + b + x
            return result

In [2]: adder(3, 5)

Out[2]: 18
```

0.1 Nested Functions

```
In [3]: def mathstuff(a, b):
            x = 10

            def multiplier(s, t):
                y = 2
                inner_result = s*t + 2
                return inner_result

            outer_result = multiplier(a, x) * b

            return outer_result

In [4]: mathstuff(2, 3)

Out[4]: 66

In [5]: def mathstuff(a, b):
            x = 10

            print("mathstuff local before:", sorted(locals().keys()))

            def multiplier(s, t):
                y = 2
                inner_result = s*t + 2
                print("\t\tmultiplier local:", sorted(locals().keys()))

                return inner_result

            outer_result = multiplier(a, x) * b

            print("mathstuff local after:", sorted(locals().keys()))

            return outer_result
```

```
In [7]: (lambda x, y: 5+ 3*x*y + 2*x + 3*y**2)(2, 4)
```

```
Out[7]: 81
```

```
In [8]: meta = 'foo bar zip zap'.split()
        meta.append
```

```
Out[8]: <function list.append>
```

```
In [9]: a = meta.append
```

```
In [10]: a('ping')
```

```
In [12]: a('pong')
```

```
In [13]: meta
```

```
Out[13]: ['foo', 'bar', 'zip', 'zap', 'ping', 'pong']
```

```
In [14]: mathstuff(2, 3)
```

```
mathstuff local before: ['a', 'b', 'x']
               multiplier local: ['inner_result', 's', 't', 'y']
mathstuff local after: ['a', 'b', 'multiplier', 'outer_result', 'x']
```

```
Out[14]: 66
```

```
In [19]: def mathstuff(a, b):
             print("mathstuff global:",
                   sorted(g for g in globals().keys()
                          if not g.startswith('_')))
             x = 10

             print("mathstuff local before:", sorted(locals().keys()))

             def multiplier(s, t):
                 print("\tmultiplier global:",
                       sorted(g for g in globals().keys()
                         if not g.startswith('_')))

                 y = 2
                 inner_result = s*t + 2
                 print("\tmultiplier local:", sorted(locals().keys()))

                 return inner_result

             outer_result = multiplier(a, x) * b

             print("mathstuff local after:", sorted(locals().keys()))

             return outer_result
```

```
In [20]: mathstuff(2, 3)
```

```
mathstuff global: ['In', 'Out', 'a', 'adder', 'exit', 'get_ipython', 'mathstuff', 'meta', 'quit']
mathstuff local before: ['a', 'b', 'x']
        multiplier global: ['In', 'Out', 'a', 'adder', 'exit', 'get_ipython', 'mathstuff', 'meta', 'quit
        multiplier local: ['inner_result', 's', 't', 'y']
mathstuff local after: ['a', 'b', 'multiplier', 'outer_result', 'x']
```

```
Out[20]: 66
```

0.2 Closures

```
In [21]: def mathstuff(a, b):
             x = a * b

             def multiplier(s, t):
                 y = 2
                 inner_result = s*t + 2 + x
                 return inner_result

             return multiplier
```

```
In [22]: import dis
         dis.dis(mathstuff)
```

```
2           0 LOAD_FAST                0 (a)
              3 LOAD_FAST                1 (b)
              6 BINARY_MULTIPLY
              7 STORE_DEREF              0 (x)

  4          10 LOAD_CLOSURE             0 (x)
             13 BUILD_TUPLE              1
             16 LOAD_CONST               1 (<code object multiplier at 0x106d084b0, file "<ipython-input
             19 LOAD_CONST               2 ('mathstuff.<locals>.multiplier')
             22 MAKE_CLOSURE             0
             25 STORE_FAST               2 (multiplier)

  9          28 LOAD_FAST                2 (multiplier)
             31 RETURN_VALUE
```

```
In [23]: m1 = mathstuff(2, 3)

In [24]: m2 = mathstuff(4, 5)

In [25]: m1

Out[25]: <function __main__.mathstuff.<locals>.multiplier>

In [26]: m2

Out[26]: <function __main__.mathstuff.<locals>.multiplier>

In [28]: m1(6, 6)

Out[28]: 44

In [29]: m2(6,6)

Out[29]: 58
```

0.3 Function Generators

```
In [30]: def genpower(p):
             def f(x):
                 return x**p
             return f

In [31]: square = genpower(2)
```

```
In [32]: square
```

```
Out[32]: <function __main__.genpower.<locals>.f>
```

```
In [33]: cube = genpower(3)
```

```
In [34]: square(6)
```

```
Out[34]: 36
```

```
In [35]: cube(6)
```

```
Out[35]: 216
```

0.4 Function Wrappers

```
In [42]: def adder(a, b):
             return a + b
```

```
In [43]: adder(3, 7)
```

```
Out[43]: 10
```

```
In [47]: def wrap_adder(a, b):
             print("Calling adder with args:", a, b)
             return adder(a, b)
```

```
In [39]: wrap_adder(3, 7)
```

```
Calling adder with args: 3 7
```

```
Out[39]: 10
```

```
In [40]: adder = wrap_adder
```

```
In [45]: dis(adder)
```

```
         ---------------------------------------------------------------------------
    TypeError                                 Traceback (most recent call last)

        <ipython-input-45-736c61610004> in <module>()
    ----> 1 dis(adder)


        TypeError: 'module' object is not callable
```

```
In [48]: dis.dis(wrap_adder)
```

```
2           0 LOAD_GLOBAL              0 (print)
              3 LOAD_CONST               1 ('Calling adder with args:')
              6 LOAD_FAST                0 (a)
              9 LOAD_FAST                1 (b)
             12 CALL_FUNCTION            3 (3 positional, 0 keyword pair)
             15 POP_TOP

  3          16 LOAD_GLOBAL              1 (adder)
             19 LOAD_FAST                0 (a)
             22 LOAD_FAST                1 (b)
             25 CALL_FUNCTION            2 (2 positional, 0 keyword pair)
             28 RETURN_VALUE
```

```
In [49]: def add_offset(x):
             return x + OFFSET

In [50]: dis.dis(add_offset)

2           0 LOAD_FAST                0 (x)
              3 LOAD_GLOBAL              0 (OFFSET)
              6 BINARY_ADD
              7 RETURN_VALUE

In [51]: add_offset(7)


         ---------------------------------------------------------------------------
    NameError                                 Traceback (most recent call last)

         <ipython-input-51-5afb09a6b5c2> in <module>()
    ----> 1 add_offset(7)


         <ipython-input-49-ace1b20d1fa3> in add_offset(x)
           1 def add_offset(x):
    ----> 2     return x + OFFSET


         NameError: name 'OFFSET' is not defined


In [52]: def adder(firstarg, secondarg):
             return firstrg + secondarg

In [53]: dis.dis(adder)

2           0 LOAD_GLOBAL              0 (firstrg)
              3 LOAD_FAST                1 (secondarg)
              6 BINARY_ADD
              7 RETURN_VALUE

In [54]: OFFSET = 10

In [55]: add_offset(7)

Out[55]: 17

In [2]: def adder(a, b):
            return a + b

In [4]: adder(3, 7)

Out[4]: 10

In [6]: def wrapped_adder(a, b):
            print("Calling func:", adder.__name__, "with args:", a, b)
            return adder(a, b)

In [7]: wrapped_adder(3, 7)

Calling func: adder with args: 3 7
```

```
Out[7]: 10

In [8]: def wrap(func):
            def wrapped(a, b):
                print("Calling func:", func.__name__, "with args:", a, b)
                return func(a, b)
            return wrapped

In [9]: a2 = wrap(adder) # generate a wrapped version of adder

In [10]: a2(3, 7)

Calling func: adder with args: 3 7

Out[10]: 10

In [11]: def sub(x, y):
             ' subtract y from x '
             return x - y

In [12]: sub(10, 2)

Out[12]: 8

In [13]: s2 = wrap(sub)

In [14]: s2(10, 2)

Calling func: sub with args: 10 2

Out[14]: 8

In [17]: def double(x):
             return 2*x

In [18]: double(3)

Out[18]: 6

In [19]: double(7)

Out[19]: 14

In [20]: d2 = wrap(double)

In [21]: d2(7)


    ---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

    <ipython-input-21-68c9ecf33e8b> in <module>()
----> 1 d2(7)


    TypeError: wrapped() missing 1 required positional argument: 'b'


In [22]: sub
```

```
Out[22]: <function __main__.sub>

In [23]: help(sub)

Help on function sub in module __main__:

sub(x, y)
    subtract y from x

In [24]: s2

Out[24]: <function __main__.wrap.<locals>.wrapped>

In [25]: a2

Out[25]: <function __main__.wrap.<locals>.wrapped>

In [26]: help(s2)

Help on function wrapped in module __main__:

wrapped(a, b)
```

0.5 Variable Arguments

```
In [27]: def poly(c0, c1, c2):
             return '{c0}+{c1}x+{c2}x^2'.format(c0=c0, c1=c1, c2=c2)

In [28]: poly(3, 6, 2)

Out[28]: '3+6x+2x^2'
```

$ 3+6x+2x^2 $

```
In [29]: def poly(c0, c1, c2, c3):
             return '{c0}+{c1}x+{c2}x^2+{c3}x^3'.format(c0=c0, c1=c1, c2=c2, c3=c3)

In [30]: poly(2, 4, 7, 5)

Out[30]: '2+4x+7x^2+5x^3'
```

$ 2+4x+7x^{2+5x}3 $

```
In [47]: def poly(cc):
             s = '{c}'.format(c=cc[0])
             if len(cc) > 1:
                 s += '+{c}x'.format(c=cc[1])
             if len(cc) > 2:
                 for i,c in enumerate(cc[2:], 2):
                     s += ' + {c}x^{i}'.format(c=c, i=i)

             return s

In [48]: poly([3])

Out[48]: '3'

In [49]: poly([3, 6])
```

```
Out[49]: '3+6x'

In [50]: poly([3, 6, 2, 8])

Out[50]: '3+6x + 2x^2 + 8x^3'

In [52]: poly([3, 7, 2, 8, 4, 9, 5])

Out[52]: '3+7x + 2x^2 + 8x^3 + 4x^4 + 9x^5 + 5x^6'

In [57]: def poly(*cc): # * unary prefix operator will bundle all positional
                        # arguments into the parameter as a tuple
             print(type(cc))
             s = '{c}'.format(c=cc[0])
             if len(cc) > 1:
                 s += ' + {c}x'.format(c=cc[1])
             if len(cc) > 2:
                 for i,c in enumerate(cc[2:], 2):
                     s += ' + {c}x^{i}'.format(c=c, i=i)

             return s

In [58]: poly(3, 7, 2, 8) # just pass those coeffs in as arguments

<class 'tuple'>

Out[58]: '3 + 7x + 2x^2 + 8x^3'

In [59]: def poly(c0, *cc): # * unary prefix operator will bundle all positional
                            # arguments into the parameter as a tuple
             print(type(cc))
             s = '{c}'.format(c=c0)
             if len(cc) > 0:
                 s += ' + {c}x'.format(c=cc[0])
             if len(cc) > 1:
                 for i,c in enumerate(cc[1:], 2):
                     s += ' + {c}x^{i}'.format(c=c, i=i)

             return s

In [60]: poly()


    ---------------------------------------------------------------------------
    TypeError                                 Traceback (most recent call last)

        <ipython-input-60-6a5cc30a4763> in <module>()
    ----> 1 poly()


        TypeError: poly() missing 1 required positional argument: 'c0'


In [61]: poly(4)

<class 'tuple'>
```

```
Out[61]: '4'

In [62]: poly(3, 6)

<class 'tuple'>

Out[62]: '3 + 6x'

In [63]: poly(3, 6, 2, 3)

<class 'tuple'>

Out[63]: '3 + 6x + 2x^2 + 3x^3'

In [64]: coeffs = (3, 6, 2, 3)

In [65]: poly(coeffs[0], coeffs[1], coeffs[2], coeffs[3])

<class 'tuple'>

Out[65]: '3 + 6x + 2x^2 + 3x^3'

In [67]: poly(*coeffs) # expand an iterable into positional arguments

<class 'tuple'>

Out[67]: '3 + 6x + 2x^2 + 3x^3'

In [68]: poly(coeffs) # this is not what we want.

<class 'tuple'>

Out[68]: '(3, 6, 2, 3)'

In [69]: from random import randint

         def numbers(n):
             return [randint(0,9) for i in range(n)]

In [70]: numbers(3)

Out[70]: [7, 8, 5]

In [71]: numbers(6)

Out[71]: [6, 5, 9, 3, 9, 5]

In [72]: a, b, c = numbers(3) # tuple unpacking

In [73]: a

Out[73]: 6

In [74]: b

Out[74]: 2

In [75]: c

Out[75]: 1
```

```
In [77]: first, second, *junk = numbers(10)

In [78]: first

Out[78]: 6

In [79]: second

Out[79]: 4

In [80]: junk

Out[80]: [1, 4, 1, 0, 6, 8, 5, 2]

In [81]: first, second, junk = numbers(10)

         ---------------------------------------------------------------------------
    ValueError                                Traceback (most recent call last)

         <ipython-input-81-a2d2f689d6d5> in <module>()
    ----> 1 first, second, junk = numbers(10)


         ValueError: too many values to unpack (expected 3)


In [83]: result = numbers(10)
          first = result[0]
          second = result[1]

In [85]: first

Out[85]: 3

In [87]: second

Out[87]: 8

In [88]: del result

In [89]: first, second, a, b, c, d, e, f, g, h = numbers(10)

In [90]: first

Out[90]: 2

In [91]: second

Out[91]: 7

In [92]: del a, b, c, d, e, f, g, h

In []:

In [93]: def message(txt, color='red', size=12, font='times'):
             print("""
         COLOR: {c}
         SIZE:  {s}
         FONT:  {f}

         {t}
             """.format(c=color, s=size, f=font, t=txt)
         )
```

```
In [94]: message("Hey, we're doing just great! Python Ninjas in 3 days!")

COLOR: red
SIZE:  12
FONT:  times

Hey, we're doing just great! Python Ninjas in 3 days!

In [96]: format = dict(color='blue', size=20, font='helvetica')

In [97]: format

Out[97]: {'color': 'blue', 'size': 20, 'font': 'helvetica'}

In [98]: # named, in order, args from dict
         message("Only 4.5 hours to go!", color=format['color'],
                                          size=format['size'],
                                          font=format['font'])

COLOR: blue
SIZE:  20
FONT:  helvetica

Only 4.5 hours to go!

In [99]: # positional
         message("Only 4.5 hours to go!", format['color'],
                                          format['size'],
                                          format['font'])

COLOR: blue
SIZE:  20
FONT:  helvetica

Only 4.5 hours to go!

In [100]: # argument reordering if named args are used
          message("Only 4.5 hours to go!", size=format['size'],
                                           color=format['color'],
                                           font=format['font'])

COLOR: blue
SIZE:  20
FONT:  helvetica

Only 4.5 hours to go!

In [101]: message("Only 4.5 hours to go!", font='goudy')

COLOR: red
SIZE:  12
FONT:  goudy

Only 4.5 hours to go!

In [102]: message('This is really cool, if you ask me', **format)
```

```
COLOR: blue
SIZE:  20
FONT:  helvetica

This is really cool, if you ask me
```

```
In [103]: message('This is really cool, if you ask me')
```

```
COLOR: red
SIZE:  12
FONT:  times

This is really cool, if you ask me
```

```
In [116]: def message(txt, color='red', size=12, font='times', **config):
              print("""
          COLOR:\t{c}
          SIZE:\t{s}
          FONT:\t{f}
          """.format(c=color, s=size, f=font))

              for k,v in config.items():
                  print("{k}:\t{v}".format(k=k.upper(), v=v))

              print("""

          {t}
              """.format(t=txt))
```

```
In [117]: format
```

```
Out[117]: {'color': 'blue',
           'align': 'center',
           'size': 20,
           'style': 'web',
           'font': 'helvetica'}
```

```
In [118]: format['align'] = 'center'
          format['style'] = 'web'
```

```
In [119]: format
```

```
Out[119]: {'color': 'blue',
           'align': 'center',
           'size': 20,
           'style': 'web',
           'font': 'helvetica'}
```

```
In [120]: message('Really flexible, but can lead to problems', **format)
```

```
COLOR:        blue
SIZE:        20
FONT:        helvetica

ALIGN:        center
STYLE:        web


Really flexible, but can lead to problems
```

```
In [128]: def wrap(func):
              def wrapper(*args, **kwargs):
                  print("Calling function:", func.__name__)
                  print("with positional arguments:", args)
                  print("and named arguments:", kwargs)
                  return func(*args, **kwargs)
              return wrapper

In [129]: def sub(x, y):
              ' subtract y from x '
              return x - y

In [130]: sub(7, 3)

Out[130]: 4

In [131]: s2 = wrap(sub)

In [132]: s2(7, 3)

Calling function: sub
with positional arguments: (7, 3)
and named arguments: {}

Out[132]: 4

In [133]: s2(7, y=4)

Calling function: sub
with positional arguments: (7,)
and named arguments: {'y': 4}

Out[133]: 3

In [134]: s2(y=5, x=22)

Calling function: sub
with positional arguments: ()
and named arguments: {'x': 22, 'y': 5}

Out[134]: 17

In [137]: from time import sleep
          def cache_func(func):

              cache = {}

              def wrapped(*args):
                  # TODO: check cache!
                  print('calling function')
                  sleep(3)
                  print('finally got result')
                  return func(*args)

              wrapped.cache = cache

              return wrapped
```

```
In [138]: sub(10, 2)

Out[138]: 8

In [139]: s2 = cache_func(sub)

In [140]: s2(10, 2)

calling function
finally got result

Out[140]: 8

In [141]: s2(12, 4)

calling function
finally got result

Out[141]: 8

In [142]: sub(12, 4)

Out[142]: 8

In [143]: s2.cache

Out[143]: {}

In [144]: sub

Out[144]: <function __main__.sub>

In [145]: help(sub)

Help on function sub in module __main__:

sub(x, y)
    subtract y from x

In [146]: s2

Out[146]: <function __main__.cache_func.<locals>.wrapped>

In [147]: help(s2)

Help on function wrapped in module __main__:

wrapped(*args)

In []:
```