Python language: Functions

The FOSSEE Group

Department of Aerospace Engineering
IIT Bombay

Mumbai, India

Outline

Functions for abstraction

- Reduce duplication of code
- Fewer lines of code: lesser scope for bugs
- Re-usability of code
- Use functions written by others, without exactly knowing how they do, what they are doing
- Enter Functions!

Defining functions

- Consider the function `f(x) = x^2`
- In Python:

```
In []: def f(x):
.....:     return x*x
.....:
In []: f(1)
In []: f(1+2j)
```

- **`def`** is a keyword
- **`f`** is the name of the function
- **`x`** the parameter of the function
- **`return`** is a keyword

Defining functions . . .

```
In [ ]: def greet():
.....:     print("Hello World!")
.....:

In [ ]: greet()
```

- `greet` is a function that takes no arguments
- It returns nothing explicitly
- Implicitly, Python returns **`None`**
- **`None`** is also a built-in, immutable data type

Defining functions . . .

```
In []: def avg(a, b):
.....:     return (a + b)/2
.....:

In []: avg(12, 10)
```

- **avg** takes two arguments
- Returns one value

Doc-strings

- It's highly recommended that all functions have documentation
- We write a doc-string along with the function definition

```
def avg(a, b):
    """Returns the average of two
    given numbers."""
    return (a + b)/2
```

```
In[]: avg?
In[]: greet?
```

Returning multiple values

- Return area and perimeter of circle, given radius
- Function needs to return two values

```
def circle(r):
    """Returns area and perimeter of
    circle given radius r"""
    pi = 3.14
    area = pi * r * r
    perimeter = 2 * pi * r
    return area, perimeter

In []: circle(4)
In []: a, p = circle(6)
```

Outline

Default arguments

```
In []: round(2.484)
In []: round(2.484, 2)

In []: s.split() # split on spaces
In []: s.split(';') # split on ';'

In []: range(10)
In []: range(1, 10)
In []: range(1, 10, 2)
```

Default arguments . . .

```
In [ ]: def welcome(greet, name="World"):
.....:     print(greet, name)
.....:

In [ ]: welcome("Hi", "Guido")
In [ ]: welcome("Hello")
```

Default arguments . . .

- Arguments with default values, should be placed at the end
- The following definition is WRONG

```
In []: def welcome(name="World", greet):
.....:     print(greet, name)
.....:
```

Keyword Arguments

```
In []: def welcome(greet, name="World"):
.....:     print(greet, name)
.....:

In []: welcome("Hello", "James")

In []: welcome("Hi", name="Guido")

In []: welcome(name="Guido",greet="Hey")

In []: welcome(name="Guido", "Hey")
```

Cannot have non-keyword args after kwargs

Built-in functions

- Variety of built-in functions are available
- **`abs, any, all, len, max, min`**
- **`pow, range, sum, type`**
- Refer here: `http://docs.python.org/library/functions.html`

Outline

Arguments are local

```
In [ ]: def change(q):
.....:     q = 10
.....:     print(q)
.....:

In [ ]: change(1)
In [ ]: print(q)
```

Variables inside function are local

```
In [ ]: n = 5
In [ ]: def change():
.....:      n = 10
.....:      print(n)
.....:
In [ ]: change()
In [ ]: print(n)
```

If not defined, look in parent scope

```
In [ ]: n = 5
In [ ]: def scope():
.....:     print(n)
.....:
In [ ]: scope()
```

- Note that **n** was not defined in **scope**
- It looked for the variable in previous scope
- See: `pythontutor.com`

global

- Use the `global` statement to assign to global variables

```
In[]: def change():
....:     global n
....:     n = 10
....:     print(n)
....:
In[]: change()
In[]: print(n)
```

Mutable variables

- Behavior is different when assigning to a list element/slice
- Python looks up for the name, from innermost scope outwards, until the name is found

```
In [ ]: name = ['Mr.', 'Steve','Gosling']
In [ ]: def change_name():
.....:     name[0] = 'Dr.'
.....:
In [ ]: change_name()
In [ ]: print(name)
```

Passing Arguments . . .

```
In [ ]: n = 5
In [ ]: def change(n):
.....:     n = 10
.....:     print(n, "inside change")
.....:
In [ ]: change(n)
In [ ]: print(n)
```

Passing Arguments . . .

```
In [ ]: name = ['Mr.', 'Steve','Gosling']
In [ ]: def change_name(x):
.....:     x[0] = 'Dr.'
.....:     print(x, 'in change')
.....:
In [ ]: change_name(name)
In [ ]: print(name)
```

Passing Arguments . . .

```
In [ ]: name = ['Mr.', 'Steve','Gosling']
In [ ]: def change_name(x):
.....:     x = [1,2,3]
.....:     print(x, 'in change')
.....:
In [ ]: change_name(name)
In [ ]: print(name)
```

Passing Arguments . . .

```
In [ ]: name = ['Mr.', 'Steve','Gosling']
In [ ]: def change_name(x):
.....:     x[:] = [1,2,3]
.....:     print(x, 'in change')
.....:
In [ ]: change_name(name)
In [ ]: print(name)
```

Outline

Functions: example

```
def signum( r ):
    """returns 0 if r is zero
    -1 if r is negative
    +1 if r is positive"""
    if r < 0:
        return -1
    elif r > 0:
        return 1
    else:
        return 0
```

Note docstrings

What does this function do?

```
def what(n):
    if n < 0: n = -n
    while n > 0:
        if n % 2 == 1:
            return False
        n /= 10
    return True
```

What does this function do?

```
def what(n):
    i = 1
    while i * i < n:
        i += 1
    return i * i == n, i
```

What does this function do?

```
def what(x, n):
    if n < 0:
        n = -n
        x = 1.0 / x

    z = 1.0
    while n > 0:
        if n % 2 == 1:
            z *= x
        x *= x
        n //= 2
    return z
```

Nested functions

```
def f(x):
    def g(x):
        return x+1
    return g(x)**2
```

Nested functions: returning functions

```
def f():
    def g(x):
        return x+1
    return g

In []: func = f()
In []: func(1)
In []: f()(1)   # Also valid!
```

Summary

- Defining functions
- Taking either 0, or more arguments
- Returning **`None`** implicitly or any number of values
- Default and keyword arguments
- Variable scope, **`global`**
- Mutable and immutable arguments
- Nested functions