Week 1: Program Design in Python

Dr Giuseppe Brandi

Northeastern University London

Outline

- Why functions
- Function signatures
- Function arguments
- Scope & lifetime of arguments
- Functions as objects

The "what" and "how" of programs

Example

```python
msg = "Hello, Joe."   # A string
send(msg, to="Joe")   # A function call
```

- Functions represent the caller's intent
- Implementation is hidden (e.g. “send with Pony Express”)

Why functions?

"Code is too *wet*. Need to make it *dry*."

- WET: Write Everything Twice
- DRY: Don't Repeat Yourself

Why functions?

What are the advantages of using functions?

- Reuse
- Readability
- Debugging
- Changeability
- Independent development

Why functions?

What are the disadvantages?

- Correctness of external library functions
- Performance of external library functions
- Dependency – sometimes unnecessary

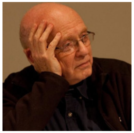
Modularisation is key principle

Programming is an art.

Phrases like “clean design” or “beautiful” have meaning.

Modularisation is key principle

Software should be organised so that dependence on information that is likely to change is restricted to a small, clearly identified, set of programs.



D. L. Parnas, CACM, June 2018

David Lorge Parnas articulated the principle of information hiding in software design with a series of articles in 1971.

Outline

Function signature

```
def name(positional arguments, keyword arguments):
    """doc-string"""
    declarations & statements
    return expression
```

First example

Write a function that converts numerical marks (%) into grades:

Mark (%)	Grade
39 and below	E
40-49	D
50-59	C
60-69	B
70 and above	A

```
>>> convert(55)
```

```
'C'
```

Functions can return at any point

```python
def convert(mark):
    """Converts a numerical mark into a grade."""
    assert not (mark < 0 or mark > 100) # Fail fast
    if mark < 40:
        return 'E'
    if mark < 50:
        return 'D'
    if mark < 60:
        return 'C'
    if mark < 70:
        return 'B'
    return 'A'
```

Functions can return multiple values

Returns minimum and maximum grade from list

```python
def stats(marks):
    minimum = min(marks)
    maximum = max(marks)
    return convert(minimum), convert(maximum)

marks = [55, 50, 75, 70, 95, 90]
# Python unpacks results for us
minimum, maximum = stats(marks)
print(f'min: {minimum}, max: {maximum}')
```

```
min:  C, max:  A
```

Functions can return objects

Tip

Returning many values from a function can confuse the caller. In such case, consider returning objects or named tuples.

Functions can return objects

```python
from collections import namedtuple

Stats = namedtuple('Stats', 'min avg max')

def stats(marks):
    return Stats(min=convert(min(marks)),
                 max=convert(max(marks)),
                 avg=convert(sum(marks)/len(marks)))

stats([55, 50, 75, 70, 95, 90])
```

```
Stats(min='C', max='A', avg='A')
```

Outline

Positional & keyword arguments

***args** and ****kwargs** are variable-length positional and keyword arguments, respectively

Generic function signature

```python
def fn(*args, **kwargs):
    # ...
    return
```

Variable-length positional arguments

- ***args** is usually used to extend required arguments
- ***args** is a tuple, so it is iterable

Generic function signature

```python
def add(x, y, *args):
    z = x + y
    for w in args:
        z += w
    return z

add(1, 2, 3, 4)  # 3, 4 are extra arguments
```

Keyword arguments

Why use keyword arguments?

- Add optional behaviour
- Extend code with backwards compatibility
- **Readability**

Optionally specify a marking scheme (other than default)

```python
def convert(mark, scheme=None):
    # ...
```

Keyword arguments

Check that a mark is within bounds

```python
def within(mark, low=0, high=100):
    return mark >= low and mark <= high

# Should a student get a distinction?
#
# Mark is a required argument.
# Override only the low value.
#
within(90, low=70)
```

Keyword arguments

In many cases, it is just a matter of clarity.

TensorFlow API example with 14 arguments

```
tf.layers.dense(inputs,
                units,
                activation=None,
                use_bias=True,
                ...
                trainable=True,
                name=None,
                reuse=None)
```

Outline

Scope & lifetime of arguments

What is the value of a b and c?

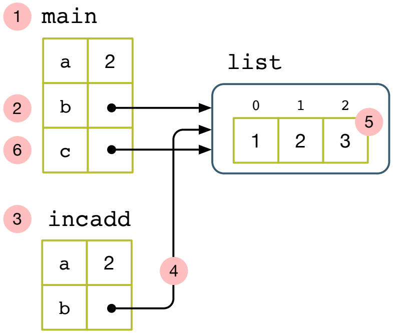
```python
def incadd(a, b):
    """Increments 'a' and appends it to 'b'."""
    a += 1
    b.append(a)
    return b

a = 2
b = [1, 2]
c = incadd(a, b)
```

Scope & lifetime of arguments

```python
def incadd(a, b):
    a += 1
    b.append(a)
    return b

a = 2
b = [1, 2]
c = incadd(a, b)
```



Scope & lifetime of arguments

Immutable variables are:

- `int`, `float`, decimal, `bool`, string, `tuple`, and `range`

Mutable variables are:

- `list`, `dict`, `set` and user-defined objects

Outline

Functions are objects

def executes at runtime – remember, there is no compile time. Python binds the function name to an object.

Functions can be arguments

```python
def wrap(fn, *args, **kwargs):
    return fn(*args, **kwargs)

wrap(convert, 55)
```

```
'C'
```

Functions are objects

def executes at runtime – remember, there is no compile time. Python binds the function name to an object.

Functions can be return values

```python
def converter():
    return convert

converter()(55)
```

```
'C'
```

Week 1: Program Design in Python

Dr Giuseppe Brandi

Scope of function arguments

Tip

A **variable scope** is the part of the program that can access a variable in its lifetime.

Scope of function arguments

An example of a closure

```python
def scaler(upper=100):
  """Converts a mark into a percentage"""
  def scale(x):  # A closure
    assert not x > upper
    return 100 * x / upper
  return scale

scale = scaler(upper=20)
convert(scale(10))
```

```
'C'
```

Summary quiz

1. What is the principle behind writing functions? Give an "elevator pitch".
2. Is this function syntactically correct?

```python
def within(low=0, mark, high=100):
    return mark >= low and mark <= high
```

3. Are these function calls different?

```python
within(5, high=10, low=0)
within(5, low=0, high=10)
```

Summary quiz

4. What is the result of the following program?[1]

```python
def incadd(a, b=[]):
    """Increments 'a' and appends it to list 'b'."""
    a += 1
    b.append(a)
    return b

a = 0
b = incadd(a)
c = incadd(a)
print(b)
print(c)
```

[1] Try to use pythontutor.com.