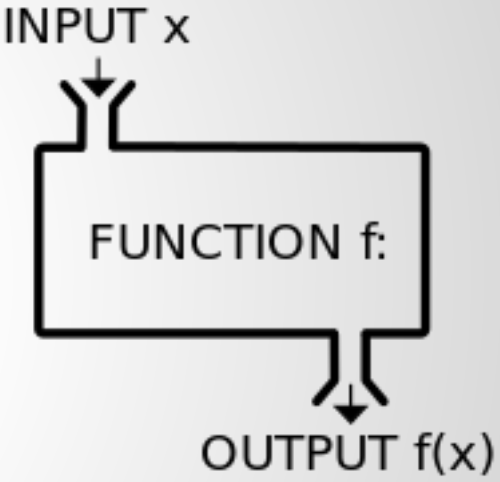
Functions

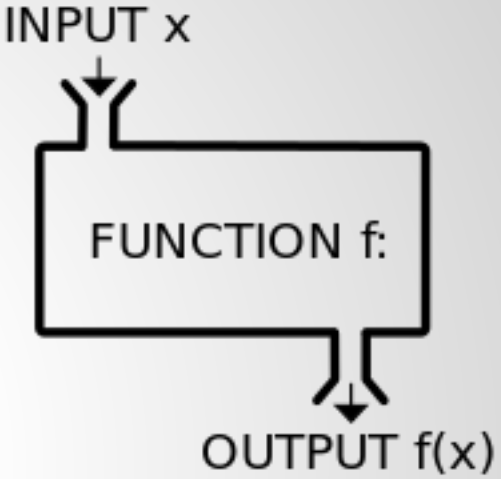


the basic element of scalable/reusable software

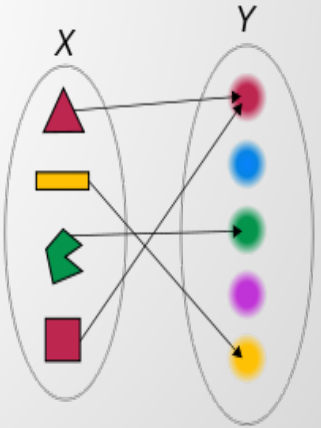
Why we need functions?

to abstract how we achieve task

to reduce repetition



Similar to but not identical to function in mathematical sense



!Manage complexity!

Difference from function (in math)

function in Python can change the environment:

- update global variable
- print smthing
- write/read/remove file

all are useful features, but can be easily abused

More terminology

function

subroutine

method

Why? (what practice shows)

building blocks to create complex systems

How to? (what is allowed/formality)

```
def functionName(params):
    statements
    return something
```

```
def PassMeArguments( the_arg1, the_arg2 ):
    return the_arg1 ** the_arg2


returned_value = PassMeArguments(10, 3)

print "returned value is", returned_value
```

How to? (what is allowed)

```
def function(param1, param2):
    return param1 + param2
```

Arguments can be lists, dictionaries, objects...

```
def ReturnSomeThings():
    value1 = 42
    value2 = 65
    return value1, value2


returned_value1, returned_value2 = ReturnSomeThings()

print "value1 is", returned_value1, "and value2 is", returned_value2
```

While Developing the Interface (Coding Style)

good NAMES

1 task - 1 function.

Avoid N tasks - 1 function

think about what you

want to achieve not HOW

check the arguments and raise exception if not valid

think about success/corner cases/ failures

While Developing the Interface (Coding Style)

- use docstrings - always… or else! doctest
- use blank lines to separate functions…
- if function has > 20 lines of code - separate into smaller
- try to have as little args as possible
- think about success/corner cases/ failures

Scope: Local vs Global

```
value1 = 103
value2 = 209

def MySimpleFunction():
    value1 = 42
    value2 = 65
    print value1, "+", value2, "=", value1 + value2

MySimpleFunction()

print "Globally, value1 is", value1, "and value2 is", value2
```

the context in which it is valid and can be used

Names are local by default unless declared to be global:

implicitly use name without defining it

explicitly use global keyword

Return statement

exits function and return the value

can be used to exit funtion earlier

Arguments/Parameters

inside the parenthesis

can have default values

either positional or with keyword

Arbitrary function arguments: avoid / refactor

Nested function (fine-tuning access control)

```
def TheOuterFunction( the_arg1 ):
    def TheInnerFunction():
        print the_arg1
    TheInnerFunction()


TheOuterFunction( "What's goin on?!?" )
```

Function itself can be an argument

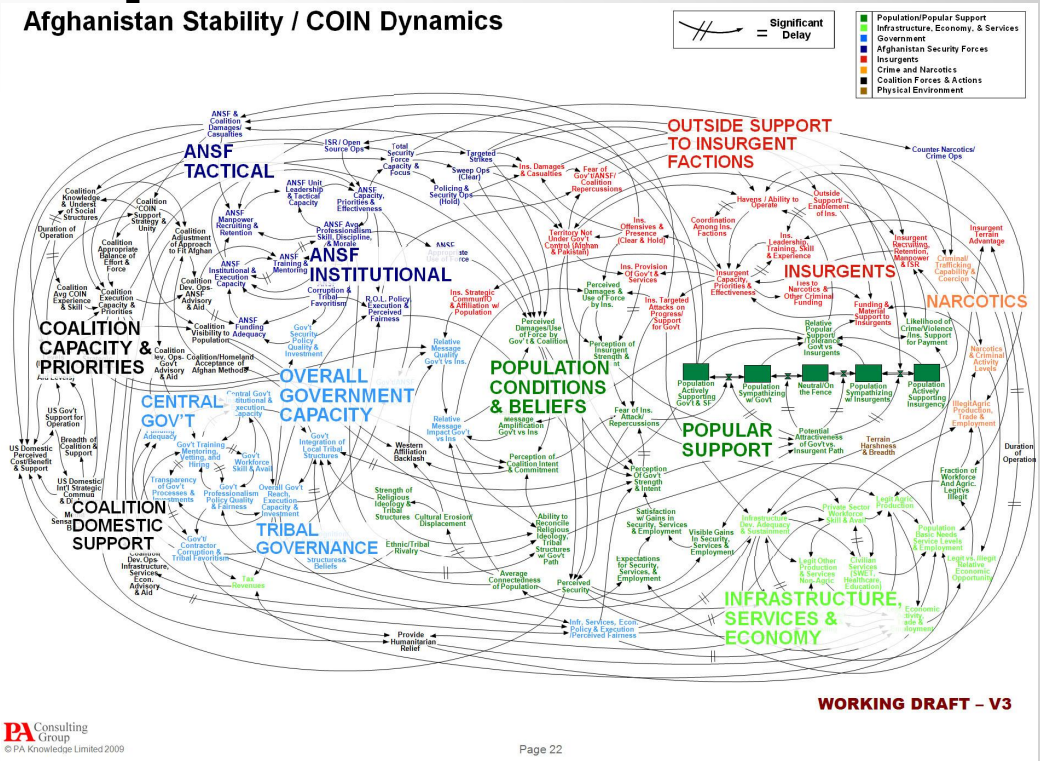
```
def ASimpleFunction():
    print "This was simple"

def AFunctionWrapper( some_function ):
    some_function()
    print "Finished running the wrapped function"


AFunctionWrapper( ASimpleFunction )
```

Side Effects: avoid if possible

communication with

external devices/resources

communication through global variables



The problem with global variables is that since every function has access to these, it becomes increasingly hard to figure out which functions actually read and write these variables.

Recursive Functions

elegant/hard to understand

function that calls itself:

★ should have base case
★ should decrease size of the problem
★ subproblems should not overlap

```
def factorial( input_integer ):
    if input_integer == 1:
        return 1
    return input_integer * factorial( input_integer - 1 )
```

Namespaces

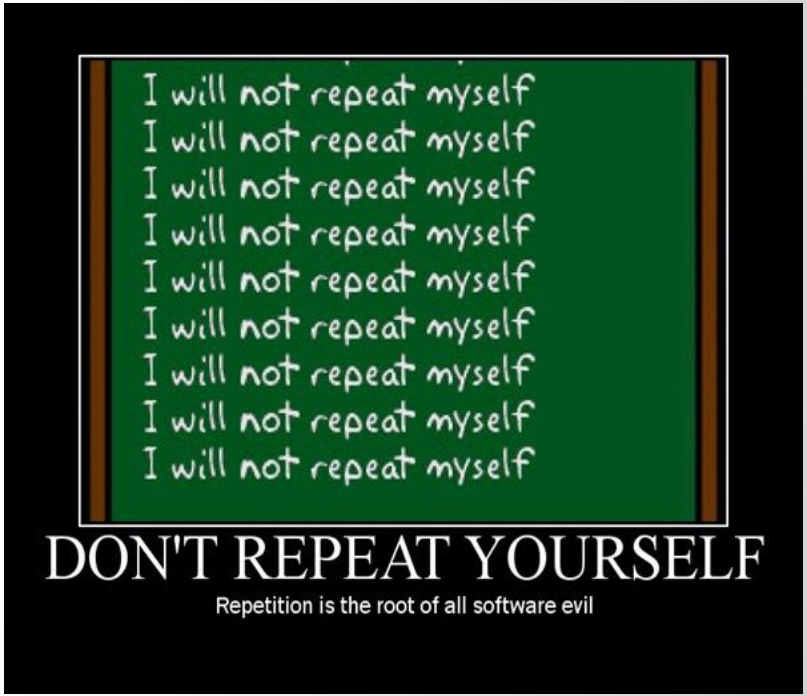
logical grouping of unique
identifiers or symbols (names)

=> avoid name clashes

well … special space for names

Benefits of Functions

- DIVIDE & CONQUER decomposition:
  from complex program
  to manageable pieces
- DRY **Don't Repeat Yourself**
  principle (removal of duplicate code)
- enabling reuse of function
- hiding implementation details



Exercise:

addPair() for two numbers and returns sum

addQuadruple() for four numbers and returns sum

addQuadruple() that can call addPair only

but cannot have + sign anywhere inside of it