# Functions

## What is a function?

- A function is a **block of code** which only runs when it is called.
- You can pass data, known as **parameters**, into a function.
- A function **can return data** as a result.

## Why using functions?

- **Maximizing code re-use and minimizing redundancy**
  - We can group operations in a single place (with a single name) and call it many times, we have to write less code.
- **Procedural decomposition**
  - Functions help you split programs into parts that have meaning. The same way making a pizza can be splitted into 'making the dough', 'adding topings', 'baking it', your programs should be split into chunks (functions), each with its sub-tasks.

## Function structure

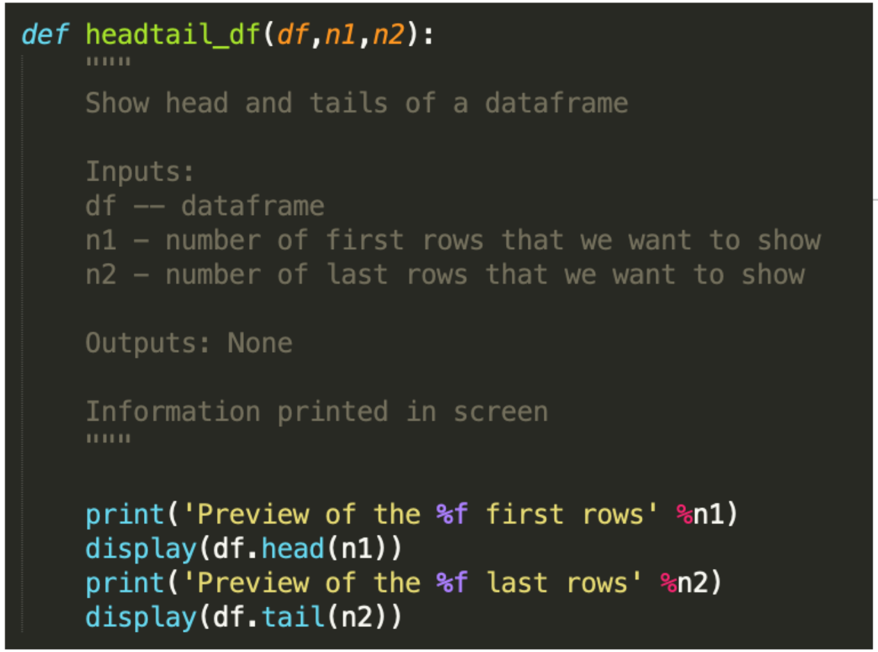
```
def name_of_function(parameters):

    '''
        Description of the function
        Input: parameters
        Output: result
        '''

    do sth (loops, conditional logic, list comprehensions, etc.)

    return result
```

## More examples



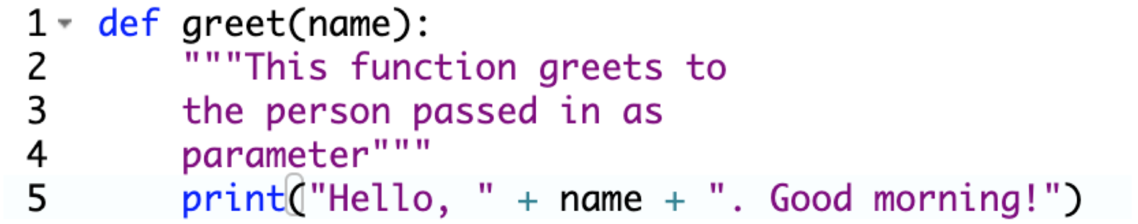
```
def headtail_df(df,n1,n2):
    """
    Show head and tails of a dataframe

    Inputs:
    df -- dataframe
    n1 - number of first rows that we want to show
    n2 - number of last rows that we want to show

    Outputs: None

    Information printed in screen
    """

    print('Preview of the %f first rows' %n1)
    display(df.head(n1))
    print('Preview of the %f last rows' %n2)
    display(df.tail(n2))
```



```
def greet(name):
        """This function greets to
        the person passed in as
        parameter"""
        print("Hello, " + name + ". Good morning!")
```

## Global vs. local variables

- **Global** variables are declared outside any function, and they can be accessed (used) on any function in the program.
- **Local** variables are declared inside a function, and can be used only inside that function.
- It is possible to have local variables with the same name in different functions. Even the name is the same, they are not the same.

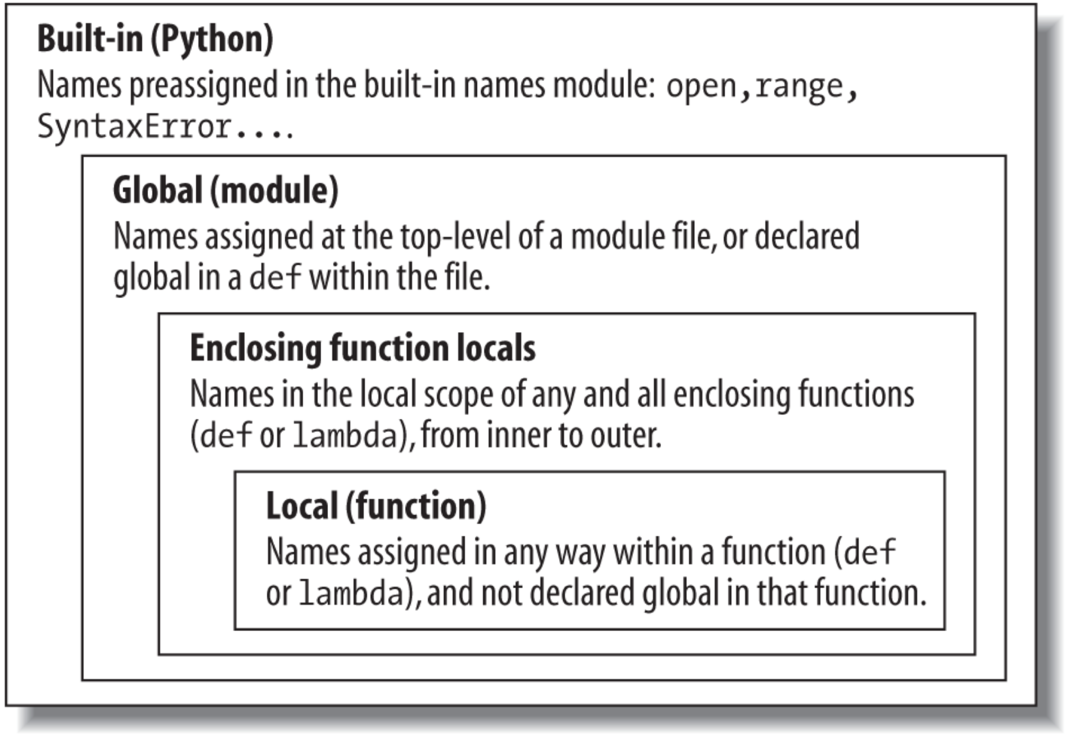
```
> a = 9

> def multiply(number, multiplier=2):
    b = number * multiplier
    return b

> c = multiply(a)

> print(a, c)
```

## Scoping rules



**Built-in (Python)**
Names preassigned in the built-in names module: `open`,`range`, `SyntaxError`....

**Global (module)**
Names assigned at the top-level of a module file, or declared global in a `def` within the file.

**Enclosing function locals**
Names in the local scope of any and all enclosing functions (`def` or `lambda`), from inner to outer.

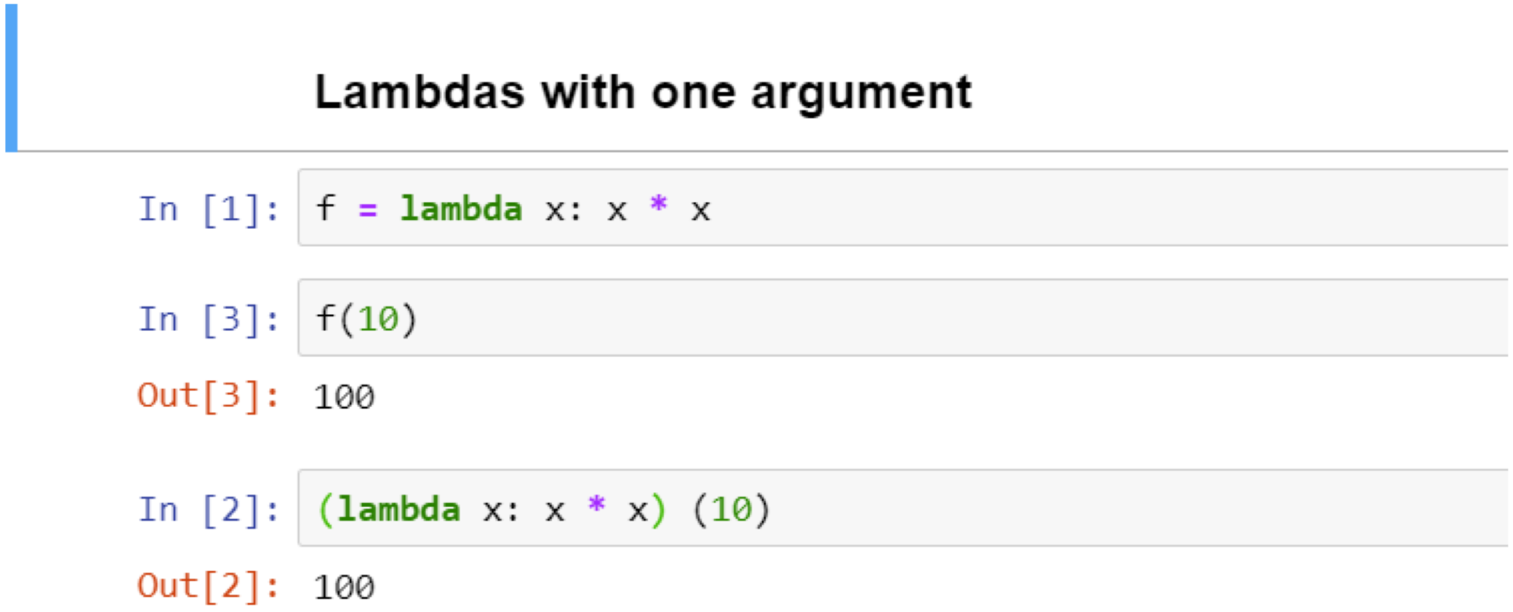
**Local (function)**
Names assigned in any way within a function (`def` or `lambda`), and not declared global in that function.

## Lambda functions

## Definition

- Function without a name (anonymous functions)
- Short expressions
- Can take multiple arguments but can only have one expression
- Any lambda function can be written as a function but not vice versa
- Possibly for short term use..(will be used once in a program?)
- Can be used within a function

## Example



### Lambdas with one argument

```
In [1]: f = lambda x: x * x

In [3]: f(10)

Out[3]: 100

In [2]: (lambda x: x * x) (10)

Out[2]: 100
```