

# Learning Python

## Getting results for beamlines and scientific programming

### 8. Basic Python: writing (and calling) functions

# Outline of topics to be covered

1. Defining functions; an example
2. Local vs. global variables
3. Function naming, pointers to functions
4. Defining parameters for a function
5. Passing parameters to a function
6. The return statement



# Defining a function

- A function is defined using the following form:

```
def FunctionName(parms...) :  
    function-statement1  
    function-statement2  
    ...  
    return <value>
```

Once this is done, the function can be used immediately in this fashion:

```
var = FunctionName(parm1, ...)
```



# A simple example of a function

- Before going over each part of the function, let's look at a simple example:

```
def factorial(n):  
    prod = 1  
    for i in range(2,n+1):  
        prod *= i  
    return prod
```

- Note that the first line of the function is indented.
  - Indentation determines when the function is completely defined.
  - Other indentation is relative to the first line of the function,
- Calling the function:

```
>>> factorial(3)  
6  
>>> factorial(9)  
362880
```

# Local vs Global variables

- Any variable ***initially defined*** inside a function is local to the function. It is deleted once the function completes. It does not affect any exterior variables that have the same name:

```
>>> prod = 0
>>> factorial(3)
6
>>> print prod
0
```

```
>>> del prod
>>> factorial(3)
6
>>> print prod
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'prod' is not defined
```



# Automatic global variables

- If a variable is used before being defined inside a function, it is assumed to be global to the function. We could define the previous function like this (note that `n` is now used without a definition):

```
def factorial():
    prod = 1
    for i in range(2,n+1):
        prod *= i
    return prod
```

```
>>> factorial()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in factorial
NameError: global name 'n' is not defined
>>> n = 4
>>> factorial()
24
```

# Creating global variables

The global statement can (but should not) be used to define a variable that can be accessed externally

```
def factorial(n):  
    global prod  
    prod = 1  
    for i in range(2,n+1):  
        prod *= i  
    return prod
```

```
>>> prod = 0  
>>> factorial(9)  
362880  
>>> print prod  
362880
```

- Use of global variables is not in general a good thing to do, since it makes for confusing code. They are sometimes OK in a module (to be discussed later) but if you find yourself wanting to use global variables, consider creating a class (also to be discussed later).



# The parts of a function in more depth: naming

```
def FunctionName(parms...):  
    <body>  
    return <value>
```

- The function name is subject to the same rules as a variable name. In fact a variable can be used to hold the name of a function

```
>>> print factorial  
<function factorial at 0x37a770>  
>>> fact = factorial  
>>> print fact  
<function factorial at 0x37a770>  
>>> fact(3)  
6
```



# The parts of a function in more depth: parameters

```
def FunctionName(parms...):
```

- The parameters for a function is most commonly a sequence of variables:

```
def Funct(p1, p2, p3):  
    return p1+10*p2+100*p3
```

- The function is then called by passing values for those parameters

```
>>> Funct(1,2,3)  
321
```

- One can also pass parameters by keyword, or even in combination, but the correct number is required

```
>>> Funct(p3=1,p1=2,p2=3)  
132  
>>> Funct(1,p3=2,p2=3)  
231  
>>> Funct(1)  
TypeError: Funct() takes exactly 3 arguments (1 given)  
>>> Funct(1,p3=2)  
TypeError: Funct() takes exactly 3 non-keyword arguments (1 given)
```

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory



# The parts of a function: optional parameters

- One can specify a default value for the final parameters for a function:

```
def Funct(p1, p2=2, p3=0):  
    return p1+10*p2+100*p3
```

- Nothing changes if values are specified for all parameters

```
>>> Funct(1,2,3)  
321
```

- But the defaults are used if a value is not specified

```
>>> Funct(1)  
21  
>>> Funct(1,p3=2)  
221
```



## Advanced topic: using arbitrary parameter lists

- It is possible to pass an adjustable number of parameters or even an arbitrary list of key word parameters to a function, but we will cover that in a later unit.



# What happens when parameters are modified?

- Changes to constant (simple) parameters are not carried outside a function

```
>>> def funct(a): a= 1    # program function definition
...
>>> b = 10
>>> funct(b)
>>> b
10
>>> funct(9)
```

- Higher-level objects are passed by reference and will be changed externally, where such changes are allowed

```
>>> def funct(a): a[0]= 1  # program function definition
...
>>> b = [0,1,2,3]
>>> funct(b)
>>> b
[1, 1, 2, 3]
>>> c = (1,2,3)
>>> funct(c)
TypeError: 'tuple' object does not support item assignment
```



## Other parts of a function: return

```
def FunctionName(parms...):  
    <body>  
    return <value>
```

- Note that use of return is optional. If the execution proceeds to the last line without encountering a return statement, the function will return the value None
- Use of return without a value also returns the value None
- One cannot return more than one value, but it is very common to return a tuple

```
    return val1, val2, val3
```

  - More clear as

```
    return (val1, val2, val3)
```



# Comments on good programming practices

- It is a very good idea to write short functions that do a single thing and then write other short functions to combine these simple functions to do more complex things
- Document the function, since you will not remember the details the next time you will want to reuse it.
- When you write a function: how will you know if it works?
  - Create test data over as wide a range of input where the function will operate as possible to test the function; use an external source (how do you know the test data is correct?)
  - For later: unit tests

Some of the most impressive programmers I have seen create the documentation and test data first and then fill in the code.



# Homework

- Write a function called `product` that accepts 2, 3 or 4 arguments and returns the product of those arguments.
- Write a function that accepts 2, 3 or 4 arguments and returns a tuple of length two where the first item is the product and the second is the sum.
  - Hint: set the default values for the optional parameters to something invalid that you can test for. (None is a great choice for that.)

