

EP305 Computer Physics

Laboratory 5

Objectives for 5th laboratory

- ❑ Getting information on functions available in `numpy`
- ❑ **User defined** functions – **parameter passing**
- ❑ Simple examples and structure of a user-defined function
- ❑ **Local** versus **global** variables
- ❑ Activity - write a number of programs that makes use of a user defined functions
 - to calculate the surface area of a cylinder, a cone, etc.
 - to convert polar to Cartesian coordinates & vice versa
 - to convert decimal degrees to deg, min, sec
- ❑ Homework – calculate the roots of a quadratic equation using two user-defined functions.
- ❑ Preparation for Homework to be submitted through Moodle

Example of a user-defined function

```

7  #----- ColorText() -----#
8  def ColorText(text, color):
9      CEND      = '\033[0m'
10     CBOLD     = '\033[1m'
11     CBLACK    = '\033[0m'
12     CRED      = '\033[31m'
13     CGREEN    = '\033[32m'
14     CYELLOW   = '\033[33m'
15     CBLUE     = '\033[34m'
16     CVIOLET   = '\033[35m'
17     CBEIGE    = '\033[36m'
18     if color == 'red':
19         return CRED + CBOLD + text + CEND
20     elif color == 'green':
21         return CGREEN + CBOLD + text + CEND
22     elif color == 'yellow':
23         return CYELLOW + text + CEND
24     elif color == 'blue':
25         return CBLUE + text + CEND
26     elif color == 'voilet':
27         return CVIOLET + text + CEND
28     elif color == 'beige':
29         return CBEIGE + text + CEND
30     else:
31         return CBLACK + text + CEND
32
33  #----- main program starts here -----#
34  print('\n') # skip a line
35
36  colour = 'red'
37  message = 'This is red'
38  print(ColorText(message, colour))
39

```

The diagram illustrates the execution flow between the `main()` function and the `ColorText()` function. It shows three sequential calls to `main()`, each of which calls `ColorText()` before returning to `main()`. The flow is represented by a blue box with arrows indicating the sequence of function calls.

This line calls the function `ColorText()`

Another example of a user-defined function

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Feb 19 18:12:41 2019
4
5  @author: Joe Bloggs (Python 3.6)
6
7  Description: for_loop_2 example: calculates values
8              function in n equally spaced steps in
9  """
10
11 import numpy as np    # needed for pi, sin(), cos()
12

```

```

13 # ----- Handling exceptions -----#
14 # Handling exceptions |
15 def newInput(message):
16     num = input(message)
17     try: # attempt to parse the number as an integer
18         num = int(num)
19         return num
20     except ValueError:
21         print(CRED + CBOLD + 'You did not enter an integer! \
22 Please try again' + CEND)
23         print('\a') # make a beep
24         return newInput(message)
25 # ----- end Handling exceptions -----#

```

```

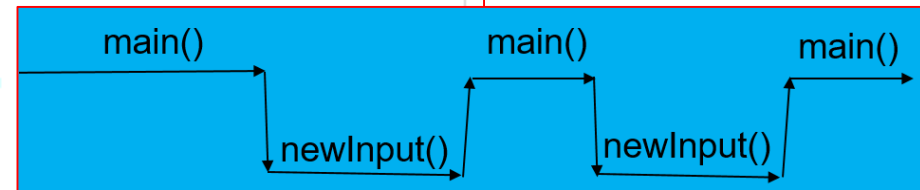
26
27 #----- main program starts here -----
28 # inform the user what is happening
29 print('\nThis programme calculates values of the sine() and cosine() function \
30 in n equally spaced steps in the range [0, 2*pi]. \
31 n is a user entered integer value')
32 #print('in the range [0, 2*pi] in n equally spaced steps')

```

```

38 # Prompt the user to enter the number of intervals required
39 message = 'Please enter an INTEGER value n (e.g. 12) : '
40 n = newInput(message) # validate the input
41

```



This line calls the function `newInput()`

Checking available functions in Numpy 2

Let us have a look at the function `np.hypot()`

In the IPython console window, type `np.hypot` then type `<Ctrl-I>`

The screenshot shows the IPython console and the help window for the `np.hypot` function. The help window is open, displaying the function's definition, type, and description. The IPython console shows the command `np.hypot` being entered, and the `Help` button is highlighted. The status bar at the bottom indicates the current line and column.

Help window:

- Object: `np.hypot`
- Function: `hypot(x1, x2[, out])`
- Type: Function
- Definition: `hypot(x1, x2[, out])`
- Type: Function
- Given the "legs" of a right triangle, return its hypotenuse.
- Equivalent to `sqrt(x1**2 + x2**2)`, element-wise. If `x1` or `x2` is scalar like (i.e., unambiguously cast-able to a scalar type), it is broadcast for use with each element of the other argument. (See Examples)
- Parameters
- `x1, x2`: array like
- Leg of the triangle(s).
- Buttons: **Help**, variable explorer

IPython console:

- Console 4/A
- In [8]: `import numpy as np`
- In [9]: `np.hypot`
- Buttons: Python console, **IPython console**, History log
- Permissions: RW
- End-of-lines: CRLF
- Encoding: UTF-8
- Line: 38
- Column: 1
- Memory: 30 %

The Help window displays information on the `np.hypot` object if it exists

Type `np.hypot` then type `<Ctrl-I>`

User-defined functions

`numpy` provides us with **the function** `hypot(x, y)`

It **receives** two parameters (x and y) and **returns** a value (`sqrt(x**2 + y**2)`)

As our first example of a **user-defined function** we will define a function which **receives** two arguments - the radius and height of a cylinder - and **returns** the volume of the cylinder.

We need a **name** for our function – we will call it **`cylinder_volume()`**

It will receive two **parameters** – the radius, r , and the height, h .

It will calculate a value equal to $\pi r^2 h$ and it will **return** this value.

defining and calling user defined functions

A Python function is a set of statements that are grouped together and named so that they can be run more than once in a program

Advantages:

- They enable code to be reused without having to be replicated in different parts of the program;
- They enable complex tasks to be broken into separate parts with each implemented by its own function.

The **def** statement defines a function, gives it a name and lists the arguments (if any) that the function expects to receive when **called**.

A function must be defined before it can be called.

We generally declare a function **before** `main()` – its **scope** is then **global** – every other function, including **main()**, knows about it.

Keyword **def**

defining a user defined functions

```
15 #
16 # define the function cylinder_vol()
17 def cylinder_vol(r_value, h_value):
18     """
19     Calculates the volume of a cylinder of radius = r_value
20     and height = h_value
21
22     Parameters
23     -----
24     r_value : float
25     DESCRIPTION.
26     h_value : float
27
28     Returns
29     -----
30     np.pi * r_value*r_value_h_value : float
31
32     """
33     return (np.pi * r_value * r_value * h_value)
```

Note colon

Parameter list

All statements in function must be indented

Keyword **return**All statements within triple quotes are the **documentation string**

Structure of function **call**

This code is in
main()

```
37  
38 # call the function cylinder_vol()  
39     c_volume = cylinder_vol(c_radius, c_height)  
40
```

1. Call the function (by its name);
2. pass the variables (`c_radius` and `c_height`) to it;
3. Assign the result to the variable `c_volume`
4. The name and variables (`c_radius`, and `c_height` and `c_volume`) **must correspond exactly** with the function definition

cylinder_volume() function example

```
1# -*- coding: utf-8 -*-
2"""
3Created on Tue Feb 19 18:12:41 2019
4
5@author: Frank Mulligan (Python 3.6)
6
7Description: returns the volume of a cylinder given its radius and height
8"""
9
10# This programme calculates the volume of a cylinder given the radius
11# and length of the cylinder
12
13import numpy as np    # needed for pi
14
15# -----#
16# define the function cylinder_vol()
17def cylinder_vol(r_value, h_value):
18    return (np.pi * r_value * r_value * h_value)
19# -----#
20
21# main program starts here
22
23# inform the user what is happening
24print('\nThis programme calculates the volume of a cylinder of radius, r, \
25and height, h. \
26\nIt uses a USER DEFINED function')
27
28ans = 'Y' # use the program at least once
29while ans == 'Y':
30    # prompt the user for the radius
31    c_radius = float(input('\nEnter the radius of the cylinder in meters \
32(e.g., 2.1) : '))
33    # and the height
34    c_height = float(input('Enter the height of the cylinder in meters \
35(e.g., 3.4) : '))
36
37
38# call the function cylinder_vol()
39c_volume = cylinder_vol(c_radius, c_height)
40
41
42# Output the results
43print('\nThe volume of the cylinder is ', \
44      '{0:>10.3f}'.format(c_volume), ' meters\b3')
45
46ans = input('\nDo you want to try another cylinder (Y/N) ')
47
```

Note colon

`r_value`, and `h_value`
are **local** variables in
`cylinder_vol()`. They are
not in **scope** outside
function `cylinder_vol()`

`c_radius` and `c_height`
are **local** variables in
`main()`. `cylinder_vol()`
does not know about them.

Output of cylinder_volume.py

```
In [15]: runfile('C:/Documents Previous computer/fmulligan/Teaching/EP305_FJM/Python/Python Progs/cylinder_volume.py', wdir='C:/Documents Previous computer/fmulligan/Teaching/EP305_FJM/Python/Python Progs')
```

This programme calculates the volume of a cylinder of radius, r , and height, h . It uses a USER DEFINED function

Enter the radius of the cylinder in meters (e.g., 2.1) : 2.1

Enter the height of the cylinder in meters (e.g., 3.4) : 3.4

The volume of the cylinder is 47.105 meters³

Do you want to try another cylinder (Y/N) N

```
In [16]:
```

Python console

IPython console

History log

Permissions: RW

End-of-lines: CRLF

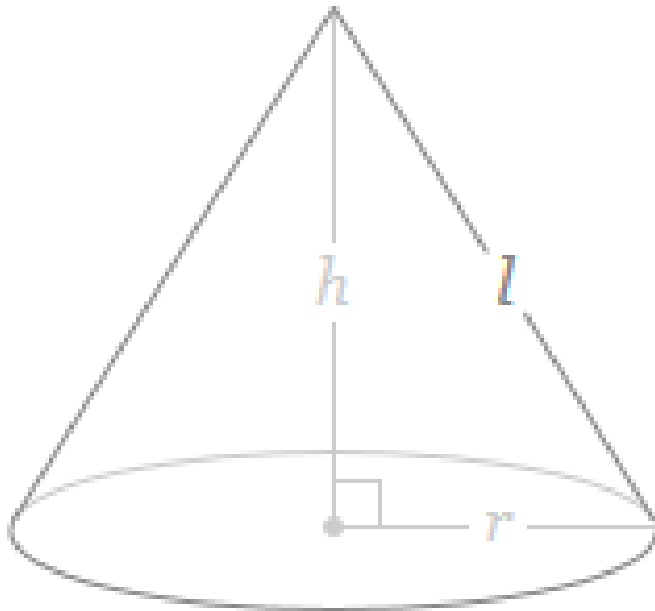
Encoding: UTF-8

Line: 33

Column: 17

Memory: 31 %

Area of a cone



$$A = \pi r l + \pi r^2$$

$$l = \sqrt{r^2 + h^2}$$

Lab Activity

- ❑ Write a program called `cone_surface_area.py`
- ❑ Which employs one simple user defined function called `cone_area()`
- ❑ to calculate the surface area (including the base area) of a cone
 - Pass the **radius** and **height** of the cone to your function
 - Return the total surface area to the `main()` program
- ❑ Print out the results, radius, height and total surface area
- ❑ Make your program “user-friendly”!

Output of cone_surface_area.py

```
This programme calculates the surface area of a cone of radius r and height h. The area includes the curved surface and the base.  
It uses a USER DEFINED function
```

```
Enter the radius of the cone base in meters      (e.g., 2.1) : 2.1
```

```
Enter the height of the cone in meters          (e.g., 3.4) : 3.4
```

```
The surface area of the cone is                40.22  meters2
```

```
Do you want to try another cone (Y/N) n
```

Lab Activity 2

- ❑ Write a program called `polar_to_Cart.py`
- ❑ Which employs one simple user defined function called `polar_2_Cart()`
- ❑ to calculate values of x and y for given values of r and θ
- ❑ Pass of r and θ to your function (θ in degrees)
 - Return the two values x , y to the `main()` program
- ❑ Print the results in a suitable format
- ❑ Make your program “user-friendly”!
- ❑ Write a program called `Cart_to_polar.py`

Lab Activity 3

- ❑ Write a program called `deg_to_dms.py`
- ❑ Which employs one simple user defined function called `deg_2_dms()`
- ❑ to calculate the angle (in degrees, minutes and seconds) given the angle in decimal degrees
 - Pass the **angle** to your function in decimal degrees
 - Return the three values **d**, **m**, **s** to the `main()` program
- ❑ Print out the angle in decimal degrees and degs, mins and seconds
- ❑ Make your program “user-friendly”!

Homework problem

- ❑ Write a program called **quadratic_roots.py**
- ❑ Which employs two simple user defined functions called **root1()** and **root2()**
- ❑ to calculate the roots of a quadratic of the form

$$Ax^2+Bx+C=0$$

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

- Pass A , B and C to each function **root1()** and **root2()**
 - Return the appropriate root to the main() program
- ❑ Print out the results in a format that makes sense!
 - ❑ Test your program on values of A , B and C that give known real roots.
 - ❑ Allow for the possibility that the roots are complex !!

`quadratic_roots.py`

Example output – real roots only

```
In [40]: runfile('C:/Documents Previous computer/fmulligan/Teaching/EP305_FJM/Python/Python Progs/Quadratic_roots1.py', wdir='C:/Documents Previous computer/fmulligan/Teaching/EP305_FJM/Python/Python Progs')
```

Programme to determine the roots of a quadratic equation of the form
 $Ax^2 + Bx + C = 0$.

It uses two USER DEFINED functions.

Enter the value of A (float): 1

Enter the value of B (float): -4.1

Enter the value of C (float): -26.6

The equation $1.0 x^2 + -4.1 x + -26.6 = 0$ has roots at
 $x = 7.600$ and $x = -3.500$

Do you want to try another equation (Y/N) N

```
In [41]: |
```

Python console

IPython console

History log

Permissions: **RW**

End-of-lines: **CRLF**

Encoding: **UTF-8**

Line: **59**

Column: **16**

Memory: **42 %**

quadratic_roots.py

Example output – complex roots

```
In [41]: runfile('C:/Documents Previous computer/fmulligan/Teaching/EP305_FJM/Python/Python Progs/Quadratic_roots1.py', wdir='C:/Documents Previous computer/fmulligan/Teaching/EP305_FJM/Python/Python Progs')
```

Programme to determine the roots of a quadratic equation of the form
 $Ax^2 + Bx + C = 0$.

It uses two USER DEFINED functions.

Enter the value of A (float): 23

Enter the value of B (float): 34

Enter the value of C (float): 56

The equation $23.0 x^2 + 34.0 x + 56.0 = 0$ has roots at
 $x = -0.739+1.374j$ and $x = -0.739-1.374j$

Do you want to try another equation (Y/N) N

```
In [42]: |
```

Python console IPython console History log

Permissions: **RW** End-of-lines: **CRLF** Encoding: **UTF-8** Line: **59** Column: **16** Memory: **42 %**

The screenshot shows a Jupyter Notebook interface. The top panel displays the documentation for the `complex` function from the `np.complex` module. The documentation includes the definition `complex(real[, imag])`, its type as a function of the builtins module, and a description: `complex(real[, imag]) -> complex number`. It also states: "Create a complex number from a real part and an optional imaginary part. This is equivalent to (real + imag*1j) where imag defaults to 0."

The bottom panel shows the IPython console with the command `In [42]: np.complex`. A dropdown menu is open, listing various attributes of the `np.complex` module, with `np.complex` selected.

Help Variable explorer

IPython console

Console 4/A

In [42]: `np.complex`

- `np.clongfloat`
- `np.column_stack`
- `np.common_type`
- `np.compare_chararrays`
- `np.compat`
- `np.complex`
- `np.complex128`
- `np.complex64`
- `np.complex_`
- `np.complexfloating`
- `np.ComplexWarning`

Python console IPython console History log

Permissions: **RW** End-of-lines: **CRLF** Encoding: **UTF-8** Line: **59** Column: **16** Memory: **40 %**

Recall
Python has
a type
`complex`.
See image
here.