Chapter 2
Python Programming for Physicists

Soon-Hyung Yook

March 31, 2017
Table of Contents I

1 Getting Started

2 Basic Programming
   • Variables and Assignments
   • Variable Types
   • Output and Input Statements
   • Arithmetic
   • Functions, Packages, and Modules
   • Built-in Functions
   • Comment Statements

3 Controlling Programs with “if” and “while”
   • The “if” statement
   • The “while” statement
   • Break and Continue

4 Lists and Arrays
   • Lists
   • Arrays
   • Reading an Array from a File
Table of Contents II

- Arithmetic with Arrays
- Slicing

User-Defined Function
Why Python?

- easy to learn
- simple to use
- enormously powerful

We will only focus on the core structure of the language for computational physics. More details are easily found from on-line documents or many text book for python.

- e.g. www.python.org
Getting Started

Code

A python program consists of a list of instructions, resembling a mixture of English words and mathematics. The list of instructions are collectively referred to as code.

Development Environment

When you are programming, you typically work in a window or windows on your computer screen that show the program you are working on and allow you to enter or edit lines of code. Such window or windows are usually called as development environment. One popular development environment for python is IDLE.(But I will not use it in this class.)

Python Version

Check the python version by typing:

python --version
Python Shell

Just type

python3

then it will show some text like:

Python 3.5.2 (default, Sep 14 2016, 11:28:32)
[GCC 6.2.1 20160901 (Red Hat 6.2.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.

The symbol “>>>” is a prompt.
First Program

- Open any text editor and type the following two-line program into the editor:
  ```python
  x=1
  print(x)
  ```
- Save the python program as “example.py” (The name of python program must end with “.py”).
- Run the following command:
  ```bash
  python3 example.py
  ```
- Also you can do the same thing in python shell.
- There are some other ways to do the same thing.
What is the program?

- instruction $\iff$ statement
- A program is a list of statements, which the computer usually executes in the order they appear in the program.
Variables and Assignments

- Quantities on interest in a program are represented by variables.
- Variables can be numbers, sets of numbers, etc.
- Variables roughly do the same role as in ordinary algebra.
- Our first example
  \[ x = 1 \]
  is an assignment statement.
- In normal algebra the variable names are usually just a single letter like \( x \).
- In many programming languages, the variable names can have two, three, or more letters.
  - Physics\_101, energy, \( x, y, z, t \)
  are examples of allowed variables.
  - 4Scores\&7Years
  is an example, which is NOT allowed as a variable name, because of &.
- In python, variable name is case sensitive.
- Naming the variable is very important!
Variable Types

- Integer
- Float
- Complex: \(1.2 + 0.6j\) Note that we use \(j\) for imaginary number!
- String

The type of a variable is set by the value that we give it.

How to change the variable type

```python
x=1
type(x)
x=1.5
type(x)
```

- Changing the variable type in a single program is not recommended.
- Good programming style: use a variable to store only one type of quantity in a given problem

see also “\(x=float(1)\)” (assign a floating point value to \(x\)), “\(x=complex(1.5)\)”.
Variable Types

String

x="This is a string"
x="1.234"

How to change a string into a number?

x="1.234"
y=float(x)

Location of the space: “x = 1” is the same with “x=1”. Space at the beginning of the line: function, loop
Output Statement

“print” statement

x=1
print(x)

Another example:

x=1
y=2
print(x,y)

One can also use like this:

x=1
y=2
print("This value of x is",x,"and the value of y is",y)

With different types of variables and with option “sep”:

x=1
z=1+3j
print(x,z,sep="...")
Input Statement

“input” statement

x=input("Enter the value of x: ")

When the computer executes the statement it does three things:

1. It prints out the quantity, if any, inside the parentheses.
2. It waits for the user to type a value on the keyboard.
3. Then the value that the user types is assigned to the variable “x”.

Example

x=input("Enter the value of x:")
print("The value of x is",x)
type(x)

Note that “x” is a string.
Input Statement

Change the string into integer, float, or complex.

```python
temp=input("Enter the value of x:")
x=float(temp)
print("The value of x is",x)
type(x)
```

or more concisely:

```python
x=float(input("Enter the value of x:"))
print("The value of x is",x)
type(x)
```

Try also this:

Enter the value of x: Hello
ValueError: invalid literal for float(): Hello
Arithmetic: Operators

Important Operators for Numerical Calculations

\begin{align*}
x &= y \quad \text{assign } y \text{ to } x \\
x &+ y \quad \text{addition} \\
x &- y \quad \text{subtraction} \\
x &\times y \quad \text{multiplication} \\
x &/ y \quad \text{division} \\
x &\times y \quad \text{raising } x \text{ to the power of } y
\end{align*}

Additional Operators for Numerical Calculations

\begin{align*}
x &\div y \quad \text{the integer part of } x \text{ divided by } y \text{ (rounded down)} \\
x &\mod y \quad \text{modulo (can be used for float, but not for complex numbers)}
\end{align*}

Another Expressions for Operators

\begin{align*}
x &+= y \quad \text{same with } x = x + y \\
x &-= y \quad \text{same with } x = x - y \\
x &*= y \quad \text{same with } x = x \times y \\
x &/= y \quad \text{same with } x = x / y \\
x &\div y \quad \text{same with } x = x \div y
\end{align*}
Operators

Comparison Operators

x==y  
x!=y  
x<y  
x<=y  
x>y  
x>=y  
in... membership

Logical Operators

x and y  
x or y  
x not y
Operators

Bitwise Operators

- \( x \lor y \) Bitwise OR
- \( x \land y \) Bitwise AND
- \( x \oplus y \) Bitwise XOR

- \( x \gg y \) shift the bits of integer \( x \) rightwards \( y \) places
- \( x \ll y \) shift the bits of integer \( x \) leftwards \( y \) places

More on the assignment operator

\( x\, y = 1, 2.5 \quad x\, y = y\, x \)

Example and Exercises in the book!
Packages, Modules, Functions

Packages

collections of related useful things

- Each package has its own name:
  - e.g. math package

Modules

- Some large packages are for convenience split into smaller subpackages
- Such subpackages are usually called as modules.
- A module in a larger package is referred to as
  packagename.modulename

Function

- Something like a black box.
- Similar to the mathematical function.
Mathematical Functions

Example of mathematical functions

log()
log10()
exp()

sin(), cos(), tan()

asin(), acos(), atan()

sinh(), cosh(), tanh()

sqrt

Usage I

import math  # import the math package
y = math.sin(math.pi)

Usage II

from math import log  # import log function from the math package
y = log(28.0)
Mathematical Functions

Usage III

from math import log, pi, sin  # import log, sin, pi from the math package

from math import *  # import everything in the math package

Import from module

import numpy.linalg  # import numpy.linalg module

from numpy.linalg import inv  # import inv function in the numpy.linalg module

See Example 2.2
Built-in Functions

There are some functions in python which do not come from any package.
These functions are always available in every program.
No need to import them.

Examples:

```python
x = float(1)
x = input("Enter the value of x: ")
print(x)
...
```

Exercises in the textbook.
Comments

Comment starts with hash mark “#”. 
The if statement

Example:

```python
x=int(input("Enter the value x= "))
if x>10:
    print("x>10")
else:
    print("x<10")
```
The if statement

Example:

```python
x=int(input("Enter the value smaller than 10: "))
if x>10:
    print("Your number is greater than 10")
    print("I will fix your number.")
x=5
print("x=",x)
```
The if statement: more elaborate example

```python
if x>10:
    print("Your number is greater than 10")
elif x>9:
    print("Your number is OK, but you’re cutting it close.")
else:
    print("Your number is fine.")
```
Example:

```python
x=int(input("Enter the value smaller than 10: "))
while x>10:
    print("Your number is greater than 10")
    x=int(input("Enter the value smaller than 10: "))
print("x=",x)
```
Stop the loop.

while x>10:
    print("the number is larger than 10. Please try again")
    x=int(input("x="))
if x==100:  # nested loop
    break

- continue: If there is continue in a loop will make the program skip the rest of the indented code in the loop and goes back to the beginning of the loop.
- The continue statement is used rather rarely in practice.

Example 2.4: Fibonacci sequence
Lists and Arrays

- In physics, we frequently use collectively a set of numbers $A = \{a_i\}$.
- Examples are a vector, matrix, and so on.
- Python provides standard features, called containers, for storing collections of numbers.
- **list, array**, dictionary, tuple, set
Lists

- List is the most basic type of container in python.
- List is a list of quantities.
- The quantities can be any type of data: integers, floats, strings, and so on.

**elements**

- The quantities in a list.
- Do not have to be all of the same type.
- However, in most of the cases, we will deal with the elements with the same type.

**Usage**

```python
r=[1,1,2,3,5,8.13,21]
print(r)
```
Lists

Usage II

```python
x=1.0
y=1.4
z=-3.1
r=[x,y,z]
print(r)
```

**NOTE:** The value of \( x \) is changed later in the program the value of \( r \) will change as well.

Change the value of element

```python
r=[1.0,1.5,-2.2]
r[1]=3.2
print(r)
```

**Note:** The index for the elements starts from zero.
Lists

Built-in Function related to list

sum, max, min, len

```python
r=[1.0,1.5,-2.2]
total=sum(r)
mean=sum(r)/len(r)
print(total)
print(mean)
```

map

- a kind of meta-function
- map creates a specialized object in the computer memory, called an iterator

```python
from math import log
r=[1.0,1.5,-2.2]
logr=list(map(log,r))
print(logr)
```
Lists

append
Add a new element to the end of the list.

r.append(2.3)

del
Delete an element by offset.

del r[-1]

- The negative index counts backward from the end of the list.
- del is not a list method (OOP), but it is a python statement.
Lists

Create an empty list

```python
r=[]
r.append(1.0)
r.append(1.4)
r.append(-3.2)
print(r)
```

**pop method**

Remove a value from the end of a list.

```python
r=[1.0,1.5,-2.2,2.6]
r.pop()
print(r)
```

- `r.pop()` is the same as `r.pop(-1)`
- pop method is similar to the python statement `del`.
- `r.pop(0)` removes the first item from the list.
Difference between lists and arrays:

1. The number of elements in an array is fixed. (One can not add or remove the elements of array.)

2. The elements of an array must all be of the same type.

3. Arrays can be two-dimensional, like matrices in algebra.
   - Contradiction: List of list is also possible:
     ```python
     a=[[1,2,3],[3,4,5],[6,3,2]]
     print(a[0][1])
     ```

4. Arrays behave roughly like vectors or matrices. (One can do some arithmetic like adding them together, etc. But lists give some error message or unexpected(?) results.)

5. Arrays work faster than lists.

6. numpy package supports the array.
Create one-dimensional array with \( n \) elements

All elements are initially equal to zero.

```
from numpy import zeros
a=zeros(4,float)
print(a)
```

Create two-dimensional array with \( n \) elements

All elements are initially equal to zero.

```
from numpy import zeros
a=zeros([3,4],float)
print(a)
```

Note that the first arguments of `zeros` in this case is itself a `list`. similar function with `zeros`: `ones`
Create one-dimensional empty array with $n$ elements

```python
from numpy import empty
a=empty(4,float)
print(a)
```

Faster than zeros or ones.

Convert list into array

```python
from numpy import array
r=[1.0,1.5,-2.2]
a=array(r,float)
print(a)
```
or

```python
from numpy import array
a=array([1.0,1.5,-2.2],float)
print(a)
```
Arrays

Covert list of list into $2 - D$ array

```python
from numpy import array
a=array([[1,2,3],[4,5,6]],float)
print(a)
```

Index of $2 - D$ array

```python
from numpy import zeros
a=zeros([2,2],float)
a[0,1]=1
a[1,0]=-1
print(a)
```
Reading an Array from a File

Use `loadtxt` function from the `numpy` package.

```
loadtxt
```

for any dimension of data!

```
from numpy import loadtxt
a=loadtxt("values.txt",float)
print(a)
```
As with lists, the individual elements of the list (array) behave like ordinary variables.

One can do arithmetic with entire arrays at once!

**Simple example**

```python
from numpy import array
a=array([1,2,3,4],int)
b=2*a
print(b)
c=a+b
print(c)
print(a+1)
print(a*b)  # not a dot product!
print(dot(a,b))
```

Matrix multiplication (and multiplication of column vector to a matrix) can be obtained by the dot function (see the example in the book).
Functions for Arrays

- Built-in functions `sum`, `max`, `min`, `len`, `map` can be applied to array as well.

**Simple example**

```python
b=array(list(map(sqrt,a)),float)
```

**size, shape methods**

```python
a=array([[1,2,3],[4,5,6]], int)
print(a.size)
print(a.shape)  # the results is tuple
```

See also the examples in the book.
Warning for Arrays

- For array, if we use \( b = a \), then the python does not make a new array \( n \).
- The direct assignment of array \( a \) to new variable \( b \) results that the \( b \) also refer the same array of numbers with \( a \), stored somewhere in the memory.

Simple example

```python
from numpy import array
a = array([1,1], int)
b = a
a[0] = 2
print(a)
print(b)
```
Warning for Arrays

copy function

To make a copy of array a use copy function

```python
from numpy import copy, array
a = array([1,1], int)
b = copy(a)
a[0] = 2
print(a)
print(b)
```
Slicing I

Slicing for lists

- Slicing works with both arrays and lists.
- Let \( r \) is a list.
- Then \( r[m:n] \) is another list composed of a subset of the elements of \( r \).
- Starting with \( m \) and going up to but not including element \( n \).

\[
\begin{align*}
\text{let } r &= [1, 3, 5, 7, 9, 11, 13, 15] \\
\text{s} &= r[2:5] \\
\text{print(s)}
\end{align*}
\]

Slicing: variants

- \( r[2:] \): all elements of the list from elements 2 up to the end of the list.
- \( r[:5] \): all elements from the start of the list up to, but not including, element 5.
- \( r[:] \): entire list.
Slicing II

Slicing for arrays

```python
from numpy import array
a = array([2, 4, 6, 8, 10, 12, 14, 16], int)
b = a[3:6]
print(b)
```

also `a[3:]`, `a[:7]`, `a[:]` work for the array.

Slicing for two-dimensional arrays

for loops

```python
r=[1,3,5,7,9,11,13,15]
for n in r:
    print(n)
    print(n*2)
print("Finished")
```

range function

- `range(5)`: generate a list [0,1,2,3,4]
- `range(2,8)`: generate a list [2,3,4,5,6,7]
- `range(2,20,3)`: generate a list [2,5,8,11,14,17]
- `range(20,2,-3)`: generate a list [20,17,14,11,8,5]

Arguments of range should be integers.
for loops II

for loop with function range

```python
for n in range(5):
    print(n**2)
```

```python
for n in range(1,11):
    print(2**n)
```

**arange function from numpy package**

- `arange(1,8,2)`: generate an array [1,3,5,7]
- `arange(1.0,8.0,2.0)`: generate an array [1.0,3.0,5.0,7.0]
- `arange(2.0,2.8,0.2)`: generate an array [2.0,2.2,2.4,2.6]

arange function can be also used with for loops.
for loops III

**linspace function from numpy package**

- linspace(2.0, 8.0, 5): divides the interval from 2.0 to 2.8 into 5 values, creating an array [2.0, 2.2, 2.4, 2.6, 2.8]
- linspace(2.0, 2.8, 3): generate a list [2.0, 2.4, 2.8]
- linspace includes the last point in the range.

linspace function can be also used with for loops.

See the Examples in the textbook.
For loops and the sum function give us two different ways to compute sums of quantities.
For loops are in general more flexible and faster than the array operation.
User-Defined Function

Simple Structure

```python
def function_name(argument1, ...):
    do something
    return x
```

Example: Factorial

```python
def factorial(n):
    f=1.0
    for k in range(1,n+1):
        f*=k
    return k
```

```python
a=factorial(10)
```
Local variables

In the above example, \( f, k \) are local variables. But if some variables are defined outside the function, those variables can be referred in the function. ⇒ Solution: Use the different files.

Function can return a List

```python
from math import cos, sin, pi

def cartesian(r, theta):
    x = r * cos(theta)
    y = r * sin(theta)
    position = [x, y]
    return position # or equivalently return [x, y]
```

```python
position_1 = cartesian(10.2, pi/4.0)
```
User-Defined Function III

Function can return an Array

```python
from math import cos, sin, pi
from numpy import array

def cartesian(r,theta):
    x=r*cos(theta)
    y=r*sin(theta)
    return array([x,y],float)

position=cartesian(10.2,pi/4.0)
```
**User-Defined Function IV**

Function can return two values

```python
from math import cos, sin, pi
from numpy import array

def cartesian(r, theta):
    x = r * cos(theta)
    y = r * sin(theta)
    return x, y

rx, ry = cartesian(10.2, pi / 4.0)
```

User-Defined Function with map

```python
def f(x):
    return 2 * x - 1

newlist = list(map(f, oldlist))
```
User-Defined Function in a Different File

```python
from mydefinitions import myfunction
```

See the examples and exercises:
Read also the rest of the chapter for good programming style.