Astronomical data analysis using Python

Lecture 2, second lecture of the course

Assumptions!!!

You are new to Python! You may or may not have programming experience in another language.

Our First Program!

In [4]:

```
a = 2
b = 5
c = a+b
d = a-b
q, r = a/b, a%b # Yes, this is allowed!

# Now, let's print!
print ("Hello World!") # just for fun
print ("Sum, Difference = ", c, d)
print ("Quotient and Remainder = ", q, r)
```

```
Hello World!
Sum, Difference =  7 -3
Quotient and Remainder =  0.4 2
```

What can we learn from this simple program?

Dynamic Typing

- We do not declare variables and types in advance. (dynamic typing)
- Variables created when first assigned values.
- Variables do not exist if not assigned. (strong typing)

Commenting

Everything after # is a comment and is ignored. Comment freely. A comment can be on its own line or following a line of code.

”print” statement

Used to print output of any kind. We will use this built-in function of Python often.

Tuple unpacking assignments

a,b = 5,6

Other Things

- Behavior of / and % operators with integer types. (changed in Python 3)
- No termination symbols at end of Python statements.

- Exception to the above...

  a = 3; b = 5

Under the Hood

- No explicit compiling/linking step. Just run... $ python First.py
- Internally, program translated into bytecode (.pyc files)
- The "translation + execution" happens line-by-line

Implications of ”line-by-line” style

- N lines will be executed before error on N+1th line halts program!
- An interactive shell.
- Interpreted language codes are generally slower than equivalent code in compiled languages.

The First Tour of the Data Types

- Numbers - Integers
- Numbers - Floats

Exploration of math module

- Strings

Methods of Declaring Strings

Concept of Sequences

Concept of Slicing

Concept of Mutability

Introduction of Object.Method concepts

Integers

In [5]:
```
8 ** 2 # Exponentiation
```

Out[5]:
```
64
```

In [6]:
```
23**200 # Auto-upgrade to "LONG INT" Notice the L!
```

Out[6]:
```
2215986975641150959165383151881728753143546002825928902065171919099670251725363088303430715834548492898142406771955476641611961977731031398212591270192026066359328531507743791619036617211088847419023131284493346710987657116681747847290261780874829638221803047530204357520 01
```

In [7]:
```
5 / 2, 5%2 # Quotient-Remainder Revisited.
```

Out[7]:
```
(2.5, 1)
```

Notice that Python is an effective scientific calculator!

Floats

In [8]:
```
5.0 * 2, 5*2.0 # Values upgraded to "higher data type".
```

Out[8]:
```
(10.0, 10.0)
```

In [9]:
```
5**0.5 # Yes, it works! Square-root.
```

Out[9]:
```
2.23606797749979
```

In [10]:
```
5 / 4.0 # No longer a quotient.
```

Out[10]:
```
1.25
```

In [11]:
```
5 % 4.0, 5 % 4.1 # Remainder, yes!!!
```

Out[11]:
```
(1.0, 0.9000000000000004)
```

Math Module

- A module can be thought of as a collection of related functions.
- To use a module,

  import ModuleName
- To use a function inside a module, simply say

  ModuleName.Function(inputs)

Let's see the math module in action!

In [12]:
```
import math
x = 60*math.pi/180.0
math.sin(x)
```

Out[12]:
```
0.8660254037844386
```

In [13]:
```
math.sin( math.radians(60) ) # nested functions
```

Out[13]:
```
0.8660254037844386
```

There are about 42 functions inside Math library! So, where can one get a quick reference of what these functions are, what they do and how to use them!?!?

In [14]:
```
print (dir(math)) # Prints all functions associated with Math module.
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acos
h', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'co
s', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'fa
ctorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'i
sclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma', 'lo
g', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi', 'pow',
'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau',
'trunc', 'ulp']
```

Help on specific functions in a module

In [15]:
```
help(math.hypot)
```

```
Help on built-in function hypot in module math:

hypot(...)
    hypot(*coordinates) -> value

    Multidimensional Euclidean distance from the origin to a point.

    Roughly equivalent to:
        sqrt(sum(x**2 for x in coordinates))

    For a two dimensional point (x, y), gives the hypotenuse
    using the Pythagorean theorem:  sqrt(x*x + y*y).

    For example, the hypotenuse of a 3/4/5 right triangle is:

        >>> hypot(3.0, 4.0)
        5.0
```

Strings

There are three methods of defining strings.

In [16]:
```
a = "John's Computer" # notice the '
```

In [17]:
```
b = 'John said, "This is my computer."' # notice the "
```

In [18]:
```
a_alt = 'John\'s Computer' # now you need the escape sequence \
```

In [19]:
```
b_alt = "John said, \"This is my computer.\"" # again escape sequence.
```

In [20]:
```
long_string = """Hello World!

I once said to people, "Learn Python!"

And then they said, "Organize a workshop!" """
```

In [21]:
```
long_string_traditional = 'Hello World! \n\nI once said to people, "Learn Pytho
n!" \
\n\nAnd then they said, "Organize an online course!" '
```

- Can be used to dynamically build scripts, both Python-based and other "languages".
- Used for documenting functions/modules. (To come later!)

String Arithmetic

In [22]:
```
s1 = "Hello" ; s2 = "World!"
```

In [23]:
```
string_sum = s1 + s2
print (string_sum)
```

```
HelloWorld!
```

In [24]:
```
string_product = s1*3
print (string_product)
```

```
HelloHelloHello
```

In [25]:
```
print (s1*3+s2)
```

```
HelloHelloHelloWorld!
```

String is a sequence!

In [26]:
```
a = "Python rocks!"
```

In [27]:
```
a[0], a[1], a[2] # Positions begin from 0 onwards.
```

Out[27]:
```
('P', 'y', 't')
```

In [28]:
```
a[-1], a[-2], a[-3] # Negative indices - count backwards!
```

Out[28]:
```
('!', 's', 'k')
```

In [29]:
```
len(a) # Measures length of both sequence/unordered collections!
```

Out[29]:
```
13
```

Sequences can be sliced!

In [30]:
```
a[2:6] # elements with indices 2,3,4,5 but not 6
```

Out[30]:
```
'thon'
```

In [31]:
```
a[8:-2] # indices 8,9 ... upto 2nd last but not including it.
```

Out[31]:
```
'ock'
```

In [32]:
```
a[:5] # Missing first index, 0 assumed.
```

Out[32]:
```
'Pytho'
```

In [33]:
```
a[5:] # Missing last index, len(a) assumed.
```

Out[33]:
```
'n rocks!'
```

Crazier Slicing

In [34]:
```
a[1:6:2],a[1],a[3],a[5] # Indices 1, 3, 5
```

Out[34]:
```
('yhn', 'y', 'h', 'n')
```

In [35]:
```
a[::2] # beginning to end
```

Out[35]:
```
'Pto ok!'
```

In [36]:
```
a[::-1] # Reverse slicing!
```

Out[36]:
```
'!skcor nohtyP'
```

In [37]:
```
a[1:6:-1] # In a[i:j:-1], changes meaning of i and j
```

Out[37]:
```
''
```

Objects and Methods - An oversimplified Introduction

An object can be thought of a construct in the memory.

It has a well defined behavior with respect to other objects. (2*3 is allowed, "a"*"b" is not!)

The properties of the object, the operations that can be performed all are pre-defined.

A method is a function bound to an object that can perform specific operations that the object supports.

```
ObjectName.MethodName(arguments)
```

OK, let's see some string methods in action!

String Methods

In [38]:
```
a = "   I am a string, I am an object, I am immutable!   "
```

In [39]:
```
a.title()
```

Out[39]:
```
'   I Am A String, I Am An Object, I Am Immutable!   '
```

In [40]:
```
a.split(",")
```

Out[40]:
```
['   I am a string', ' I am an object', ' I am immutable!   ']
```

In [41]:
```
a.strip() # Remove trailing and leading whitespaces.
```

Out[41]:
```
'I am a string, I am an object, I am immutable!'
```

Strings are Immutable!

In [42]:
```
print (a) # Check the value!
```

```
I am a string, I am an object, I am immutable!
```

In [43]:
```
a.title() # Transform string to title case ... really?
```

Out[43]:
```
'   I Am A String, I Am An Object, I Am Immutable!   '
```

In [44]:
```
print (a) # Nothing changed! Strings are immutabe.
```

```
I am a string, I am an object, I am immutable!
```

In [45]:
```
b = a.title() # String methods return strings instead.
```

In [46]:
```
print (b)
```

```
I Am A String, I Am An Object, I Am Immutable!
```

In [47]:
```
a[3] = "x" # Immutability implies no in-place changes.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-47-1c6b97054996> in <module>
----> 1 a[3] = "x" # Immutability implies no in-place changes.

TypeError: 'str' object does not support item assignment
```

Getting Help

In [48]:
```
print (dir(a)) # a is a string object.
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnew
args__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '_
_le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__red
uce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '_
_sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center',
'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifie
r', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'j
oin', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'r
emovesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit',
'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
```

In [49]:
```
help(a.find)
```

```
Help on built-in function find:

find(...) method of builtins.str instance
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```