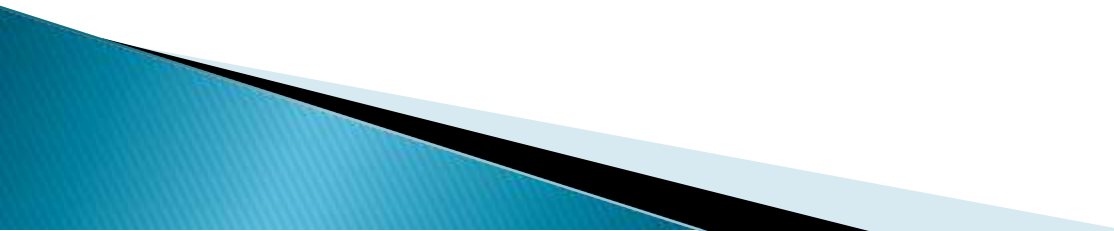


# Python: An Introduction

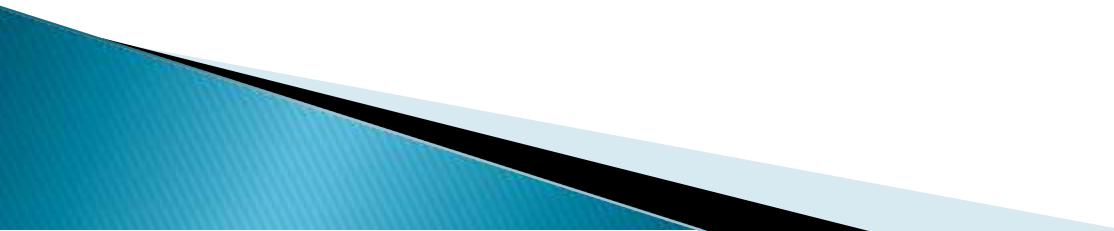
Presented By Prasenjit Das



# Agenda

- ▶ Introduction
  - ▶ Running Python
  - ▶ Python Programming
    - Data types
    - Control flows
    - Classes, functions
- 

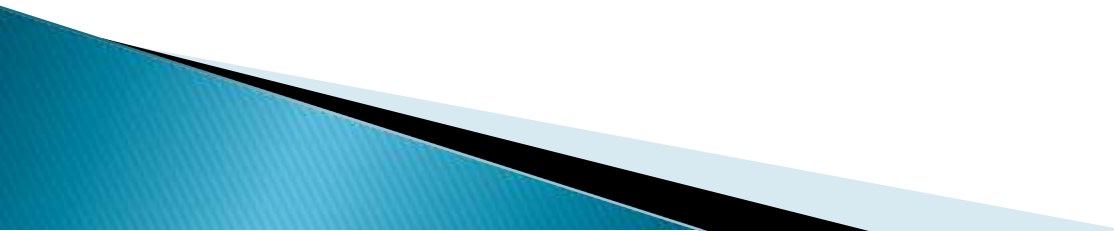
# What is python?

- ▶ Object oriented language
  - ▶ Interpreted language
  - ▶ Supports dynamic data type
  - ▶ Independent from platforms
  - ▶ Focused on development time
  - ▶ Simple and easy grammar
  - ▶ Automatic memory management
  - ▶ It's free (open source)!
- 

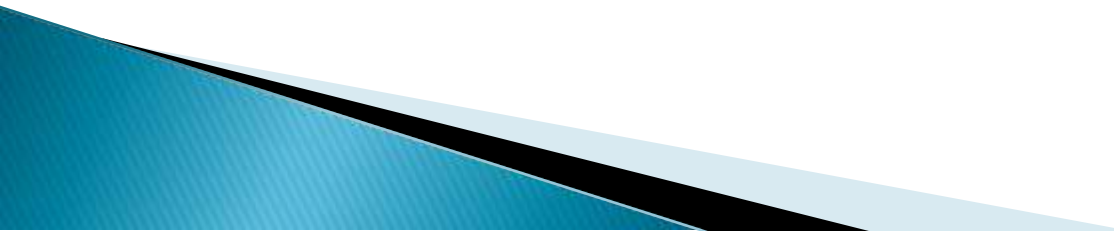
# Timeline

- ▶ Python born, name picked – Dec 1989
  - By Guido van Rossum, now at GOOGLE
- ▶ First public release (USENET) – Feb 1991
- ▶ python.org website – 1996 or 1997
- ▶ 2.0 released – 2000
- ▶ Python Software Foundation – 2001
- ▶ ...
- ▶ 2.4 released – 2004
- ▶ 2.5 released – 2006
- ▶ Current version: 3.1.1

# Language properties

- ▶ Everything is an object
  - ▶ Modules, classes, functions
  - ▶ Exception handling
  - ▶ polymorphism
  - ▶ Static scoping
  - ▶ Indentation for block structure
- 

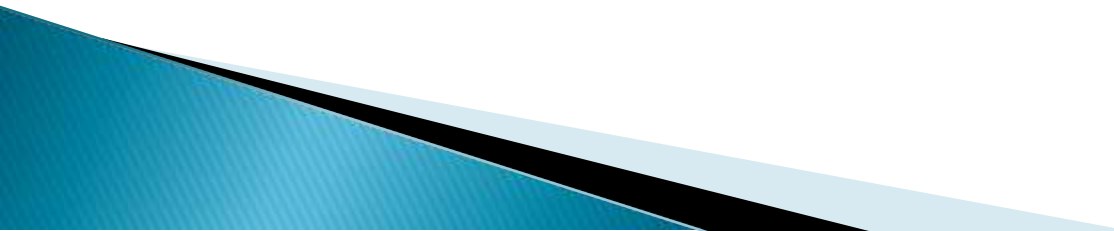
# High-level data types

- ▶ Numbers: int, long, float, complex
  - ▶ Strings: immutable
  - ▶ Lists and dictionaries
  - ▶ Extension modules can define new “built-in” data types
- 

# Why learn python?

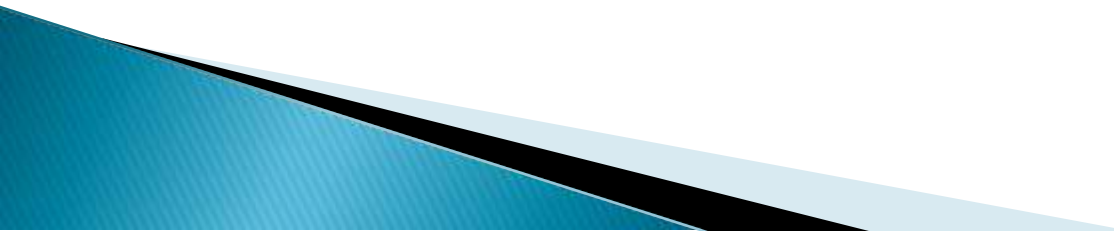
- ▶ Fun-to-use "Scripting language"
- ▶ Object-oriented
  - Highly educational
- ▶ Very easy to learn
- ▶ Powerful, scalable, easy to maintain
  - high productivity
  - Lots of libraries

# Why learn python? (cont.)

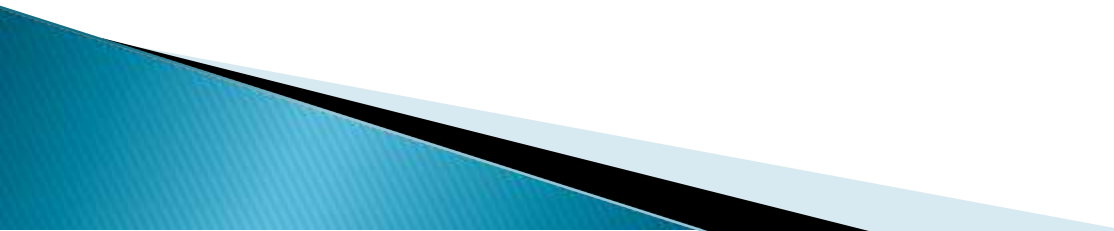
- ▶ Reduce development time
  - ▶ Reduce code length
  - ▶ Easy to learn and use as developers
  - ▶ Easy to understand codes
  - ▶ Easy to do team projects
  - ▶ Easy to extend to other languages
- 



# Where to use python?

- ▶ System management (i.e., scripting)
  - ▶ Graphic User Interface (GUI)
  - ▶ Internet programming
  - ▶ Database (DB) programming
  - ▶ Text data processing
  - ▶ Distributed processing
  - ▶ Numerical operations
  - ▶ Graphics
  - ▶ And so on...
- 

# Python vs. Java

- Code 5–10 times more concise
  - Dynamic typing
  - Much quicker development
    - no compilation phase
    - less typing
  - Yes, it runs slower
    - but development is so much faster!
  - Similar (but more so) for C/C++
  - ▶ Use Python with Java: JPython!
- 

# Running Python Interactively

- ▶ Start python by typing "python"
  - /afs/isis/pkg/isis/bin/python
- ▶ Comments start with '#'
  - `>>> 2+2 #Comment on the same line as text`
  - `4`
  - `>>> 7/3 #Numbers are integers by default`
  - `2`
  - `>>> x = y = z = 0 #Multiple assigns at once`
  - `>>> z`
  - `0`

# File naming extension

- ▶ python files usually end with the suffix `.py`
- ▶ but executable files usually don't have the `.py` extension

# Comments

- ▶ Start with # and go to end of line
- ▶ What about C, C++ style comments?
  - NOT supported!

# Python Syntax

- ▶ Much of it is similar to C syntax
- ▶ Exceptions:
  - missing operators: `++`, `--`
  - no curly brackets, `{ }`, for blocks; uses **whitespace**
  - different keywords
  - lots of extra features
  - **no type declarations!**

# Simple data types

- ▶ Numbers
  - Integer, floating–point, complex!
- ▶ Strings
  - characters are strings of length 1
- ▶ Booleans are **False** or **True**

# Numbers

- ▶ The usual notations and operators
  - 12, 3.14, 0xFF, 0377,  $(-1+2)*3/4**5$ , `abs(x)`,  $0 < x \leq 5$
  
- ▶ Integer division truncates :
  - $1/2 \rightarrow 0$       # `float(1)/2`  $\rightarrow 0.5$



# Strings and formatting

```
i = 10
```

```
d = 3.1415926
```

```
s = "I am a string!"
```

```
print "%d\t%f\t%s" % (i, d, s)
```

```
print "newline\n"
```

```
print "no newline"
```

# Variables

- ▶ No need to declare
- ▶ Need to assign (initialize)
  - use of uninitialized variable raises exception
- ▶ Not typed

```
if friendly: greeting = "hello world"
else: greeting = 12**2
print greeting
```
- ▶ ***Everything*** is a variable:
  - functions, modules, classes

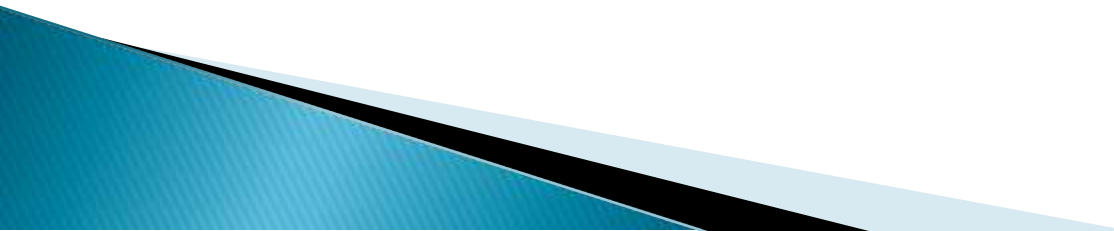
# Simple data types: operators

- ▶ `+` `-` `*` `/` `%` (like C)
  - ▶ `+=` `-=` etc. (no `++` or `--`)
  - ▶ Assignment using `=`
    - but semantics are different!
- ```
a = 1  
a = "Prasenjit" # OK
```
- ▶ Can also use `+` to concatenate strings

# Strings

- "hello"+"world" "helloworld" # concatenation
- "hello"\*3 "hellohellohello" # repetition
- "hello"[0] "h" # indexing
- "hello"[-1] "o" # (from end)
- "hello"[1:4] "ello" # slicing
- len("hello") 5 # size
- "e" in "hello" 1 # search
- New line: "escapes: \n "
- Line continuation: triple quotes '''
- Quotes: 'single quotes', "raw strings"

# Methods in string

- ▶ upper()
  - ▶ lower()
  - ▶ capitalize()
  - ▶ count(s)
  - ▶ index(s)
- 

# Compound Data Type: List

## ▶ List:

- Lists are used to store multiple items in a single variable.
- Lists are created using square brackets:

```
thislist = ["apple", "banana", "cherry"]  
print(thislist)
```

List items are ordered, changeable, and allow duplicate values

# Dictionaries

- ▶ Dictionaries are used to store data values in key:value pairs.
- ▶ A dictionary is a collection which is ordered\*, changeable and do not allow duplicates.
- ▶ As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*
- ▶ 

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
print(thisdict)
```

# Tuples

- ▶ Tuples are used to store multiple items in a single variable.
- ▶ Tuple items are ordered, unchangeable, and allow duplicate values.
- ▶ Tuples are written with round brackets.

```
thistuple = ("apple", "banana", "cherry")  
print(thistuple)
```

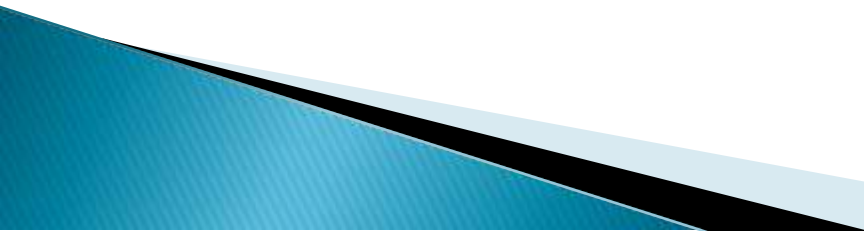


E.g.,

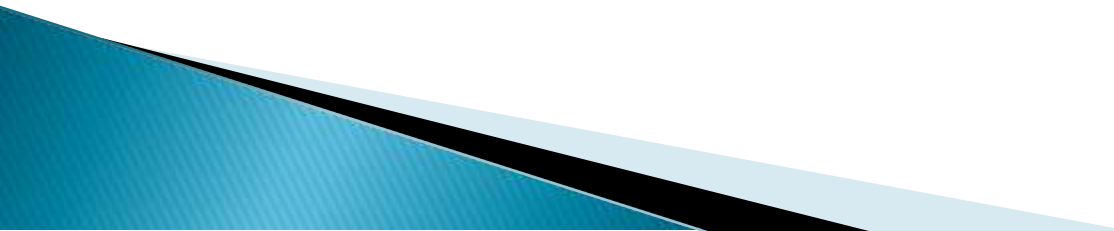
```
>>> t = ()  
>>> t = (1, 2, 3)  
>>> t = (1, )  
>>> t = 1,  
>>> a = (1, 2, 3, 4, 5)  
>>> print a[1] # 2
```



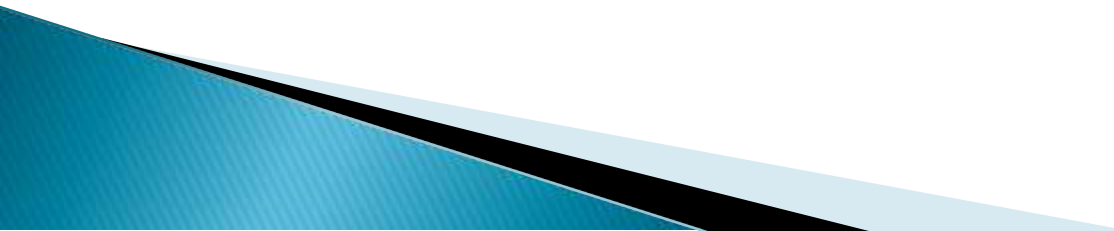
# List vs. Tuple

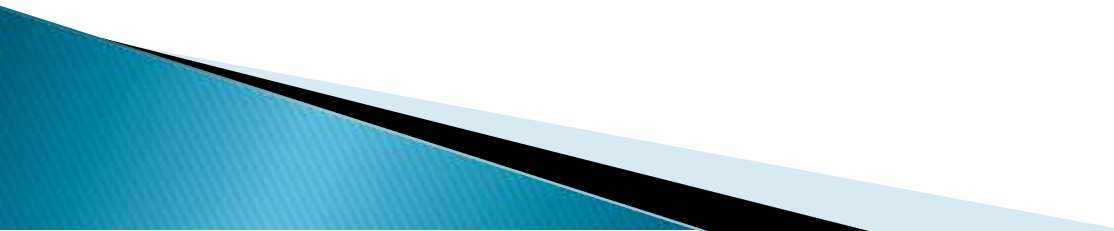
- ▶ What are common characteristics?
    - Both store arbitrary data objects
    - Both are of sequence data type
  - ▶ What are differences?
    - Tuple **doesn't allow modification**
    - Tuple doesn't have methods
    - Tuple supports format strings
    - Tuple supports variable length parameter in function call.
    - Tuples **slightly faster**
- 

# Data Type Wrap Up

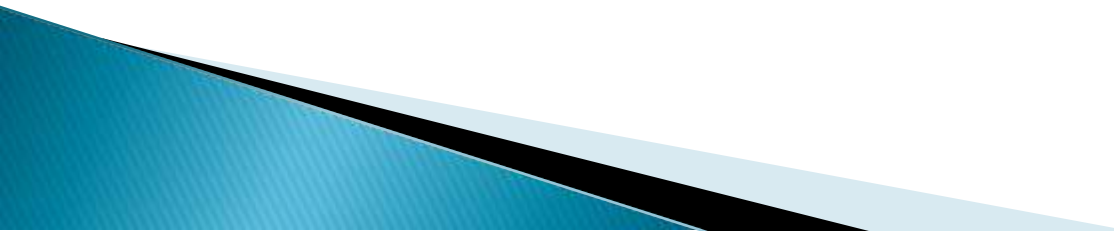
- ▶ Integers: 2323, 3234L
  - ▶ Floating Point: 32.3, 3.1E2
  - ▶ Complex: 3 + 2j, 1j
  - ▶ Lists: l = [1,2,3]
  - ▶ Tuples: t = (1,2,3)
  - ▶ Dictionaries: d = {'hello' : 'there', 2 : 15}
- 

# Input

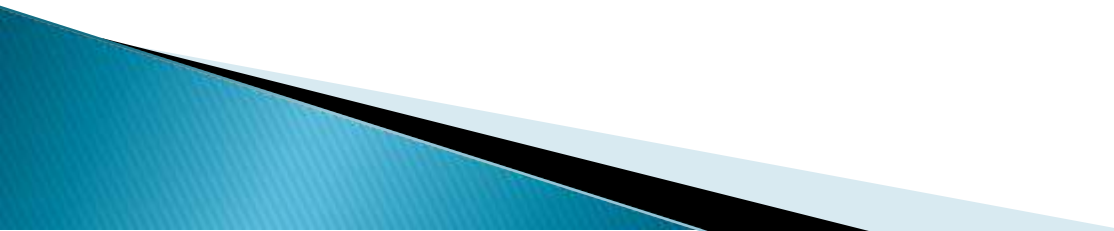
- ▶ Python allows for user input.
  - ▶ That means we are able to ask the user for input.
  - ▶ The method is a bit different in Python 3.6 than Python 2.7.
  - ▶ Python 3.6 & Upper versions uses the `input()` method.
  - ▶ Python 2.7 uses the `raw_input()` method.
- 

- ▶ Python 3.6
  - ▶ `username = input("Enter username:")`  
`print("Username is: " + username)`
  - ▶ Python 2.7
  - ▶ `username = raw_input("Enter username:")`  
`print("Username is: " + username)`
- 

# OOP Terminology

- ▶ **class** -- a template for building objects
  - ▶ **instance** -- an object created from the template (an instance of the class)
  - ▶ **method** -- a function that is part of the object and acts on instances directly
  - ▶ **constructor** -- special "method" that creates new instances
- 

# Control flow

- ▶ Python supports the usual logical conditions from mathematics:
  - ▶ Equals:  $a == b$
  - ▶ Not Equals:  $a != b$
  - ▶ Less than:  $a < b$
  - ▶ Less than or equal to:  $a \leq b$
  - ▶ Greater than:  $a > b$
  - ▶ Greater than or equal to:  $a \geq b$
  - ▶ These conditions can be used in several ways, most commonly in "if statements" and loops.
  - ▶ An "if statement" is written by using the if keyword.
- 

# Control flow

```
▶ a = 200
  b = 33
  if b > a:
    print("b is greater than a")
  elif a == b:
    print("a and b are equal")
  else:
    print("a is greater than b")
```


# Control flow

- ▶ Python has two primitive loop commands:
- ▶ while loops
- ▶ for loops
- ▶ #while loop
- ▶ Print i as long as i is less than 6:
- ▶ 

```
i = 1
while i < 6:
    print(i)
    i += 1
```



# Control flow

- ▶ A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
  - ▶ With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.
- 

# Control flow

- ▶ Print each fruit in a fruit list:
- ▶ #for loop
- ▶ 

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)
```

# Control flow: odds & ends

- ▶ **continue** statement like in C

- ▶ **pass** keyword:

```
if a == 0:
```

```
    pass # do nothing
```

```
else:
```

```
    # whatever
```

# Defining functions

```
def foo(x):  
    y = 10 * x + 2  
    return y
```

- ▶ All variables are local inside function
- ▶ Arguments passed by **value**

# Executing functions

```
def foo(x):  
    y = 10 * x + 2  
    return y  
  
print foo(10)    # 102
```

# Python: Pros & Cons

## ▶ Pros

- **Free** availability (like Perl, Python is open source).
- **Stability** (Python is in release 3.11).
- Very **easy** to learn and use
- Good **support** for objects, modules, and other reusability mechanisms.
- Easy integration with and **extensibility** using C and Java.

## ▶ Cons

- Smaller pool of Python developers compared to other languages, such as Java
- Software **performance** slow, not suitable for high performance applications

Thank you

