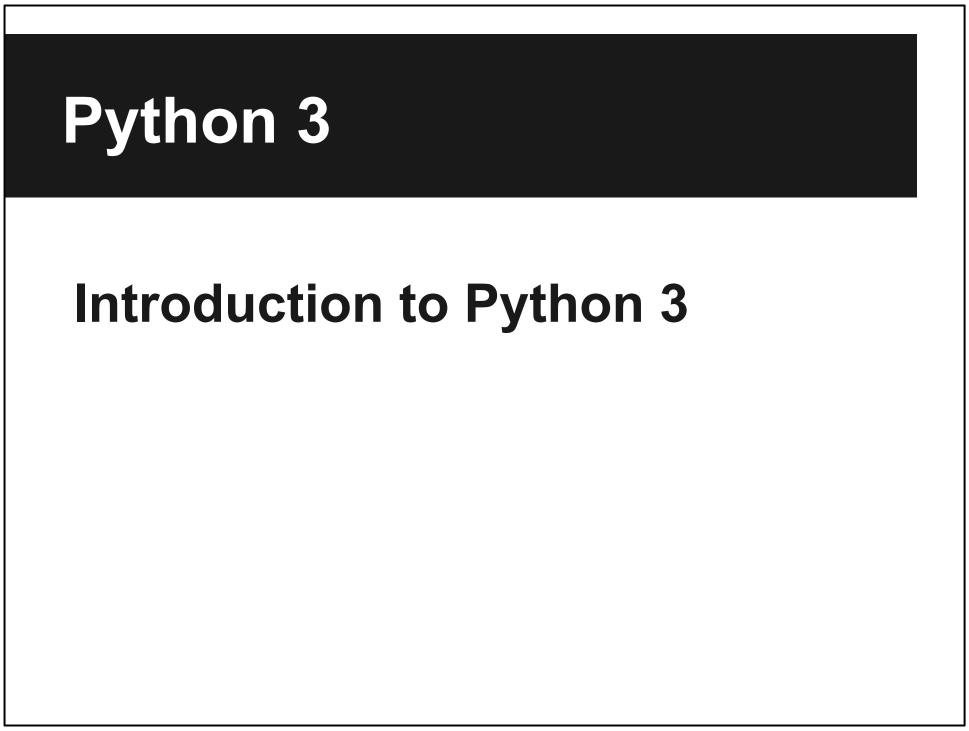


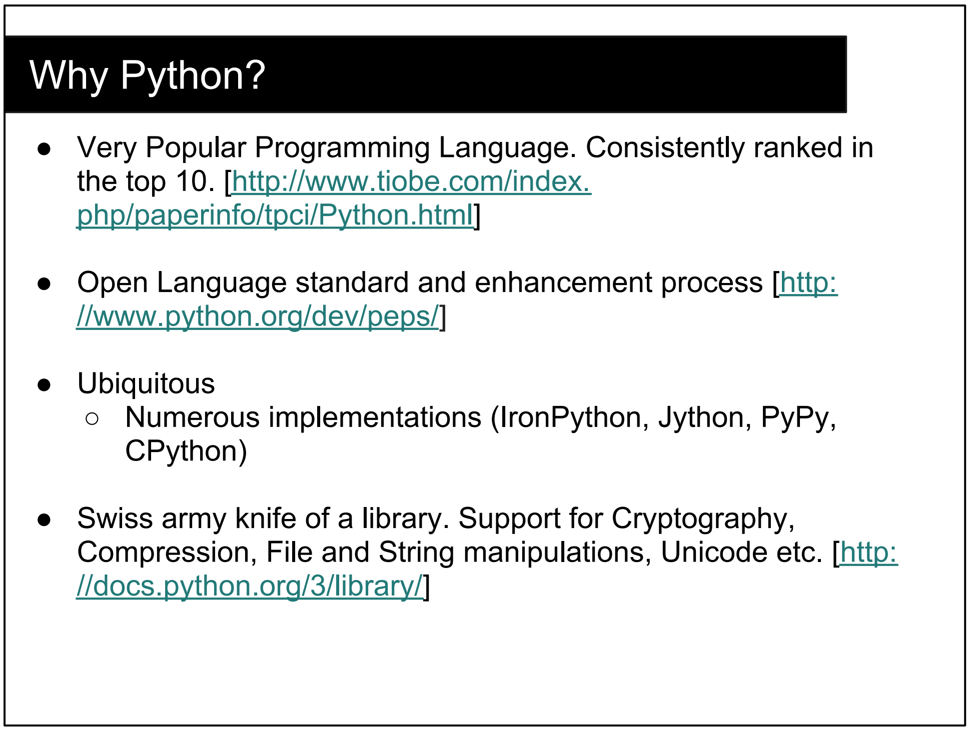
# Python 3

## Introduction to Python 3

Welcome class. My name is Subrat Sharma and  I am a CAETE student.

For my class presentation I will be talking about Python 3 - just a brief history of the programming language and it's new direction.

Please feel free to stop me anytime if you have any questions.



## Why Python?

- Very Popular Programming Language. Consistently ranked in the top 10. [http://www.tiobe.com/index.php/paperinfo/tpci/Python.html]
- Open Language standard and enhancement process [http://www.python.org/dev/peps/]
- Ubiquitous
  - Numerous implementations (IronPython, Jython, PyPy, CPython)
- Swiss army knife of a library. Support for Cryptography, Compression, File and String manipulations, Unicode etc. [http://docs.python.org/3/library/]

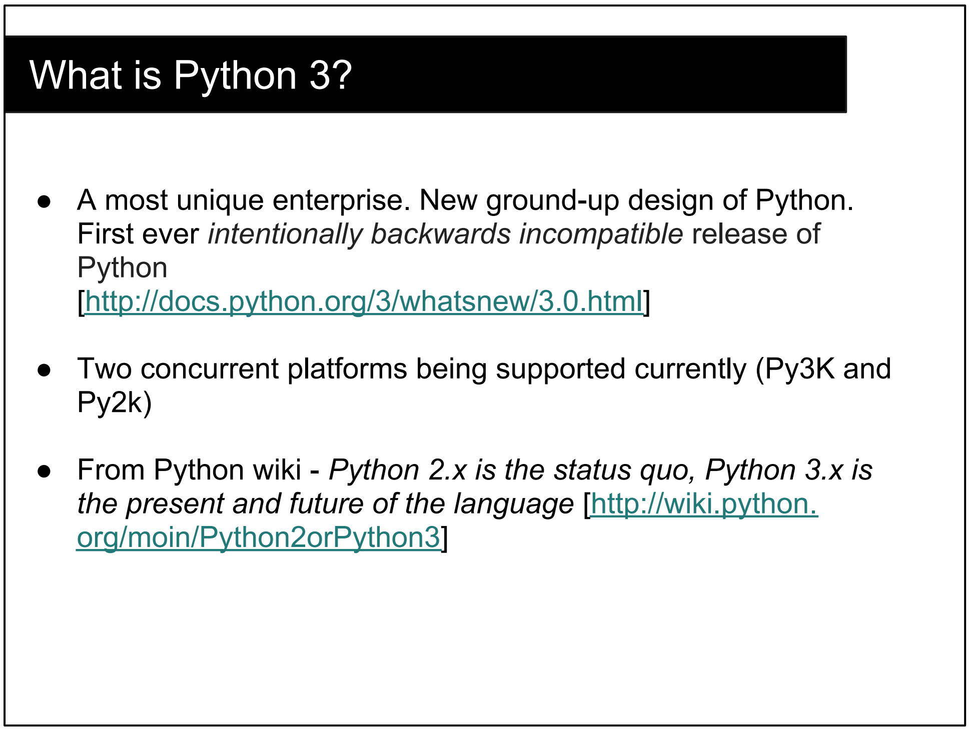
**Why Python?**

The question here being why Python is relevant to us as Software Engineers. Python is one of the most popular computer programming languages published. It is consistently ranked in the top 10 according to the TIOBE Software.

The Python language standard is very well defined and has a very open process for introducing enhancements. The process of introducing enhancements is called Python Enhancement Proposals.

Ubiquitous - If you look under the hood of any modern operating system. Python will without fail be acting as a glue for tying different parts of the system together. Python is one of the very few languages that are available in a variety of implementations across a different platforms. To name a few. CPython, PyPy, IronPython etc. This availability across varied platforms makes it a truly write once run everywhere language.

Python boasts one of the most diverse set of libraries of any programming languages. Including string manipulation, debugging, compression, cryptography, unicode etc. to name a few.



## What is Python 3?

- A most unique enterprise. New ground-up design of Python. First ever *intentionally backwards incompatible* release of Python [http://docs.python.org/3/whatsnew/3.0.html]
- Two concurrent platforms being supported currently (Py3K and Py2k)
- From Python wiki - *Python 2.x is the status quo, Python 3.x is the present and future of the language* [http://wiki.python.org/moin/Python2orPython3]

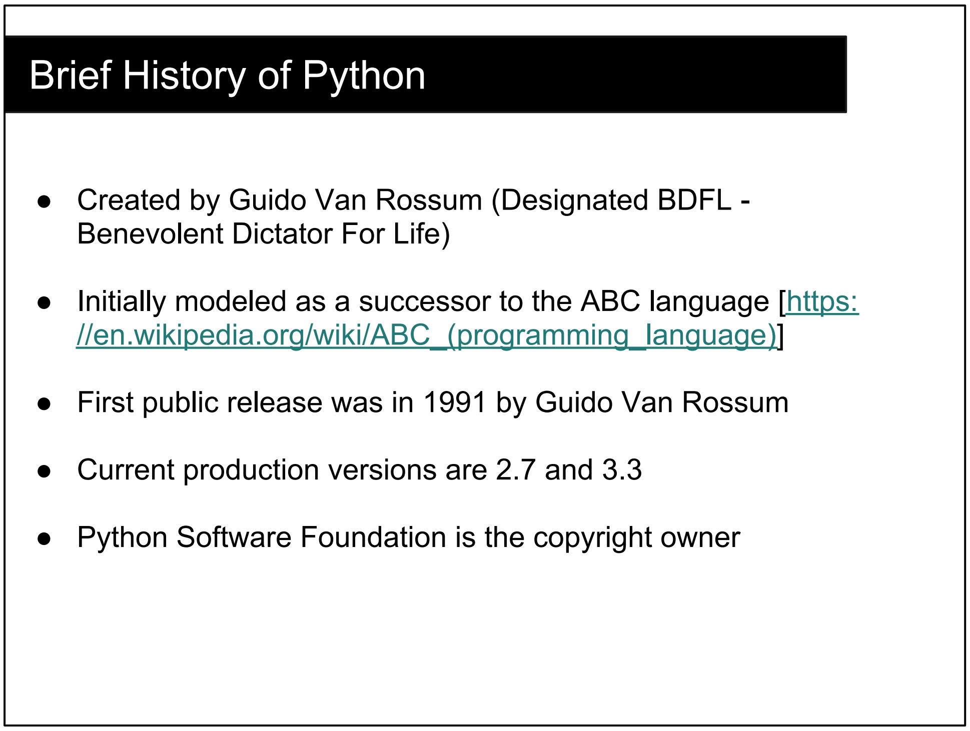
**What is Python 3?**

[A most unique enterprise] As is common with any software that has a published API; the curse of backwards compatibility also befell Python. However, overtime the language designers felt that the language improvement started to suffer because of the design decisions made decades ago.

Different from the common practice, they decided to break the need to be backwards compatible. This has enabled making huge improvements to the language possible. A rather large problem however still remains; all code that was written to work on earlier versions of Python (<2.6) need to be ported if new features from Python 3 are to be used.

[For us software engineers] This could be a hint at future market demands

The downside of this approach of language furtherment. There are concurrently two versions of Python being actively developed. There is a greater focus towards making the future of the language better. The following is a quote directly from the Python wiki. *Python 2.x is the status quo, Python 3.x is the present and future of the language*.



## Brief History of Python

- Created by Guido Van Rossum (Designated BDFL - Benevolent Dictator For Life)
- Initially modeled as a successor to the ABC language [https://en.wikipedia.org/wiki/ABC_(programming_language)]
- First public release was in 1991 by Guido Van Rossum
- Current production versions are 2.7 and 3.3
- Python Software Foundation is the copyright owner

**Brief History of Python**

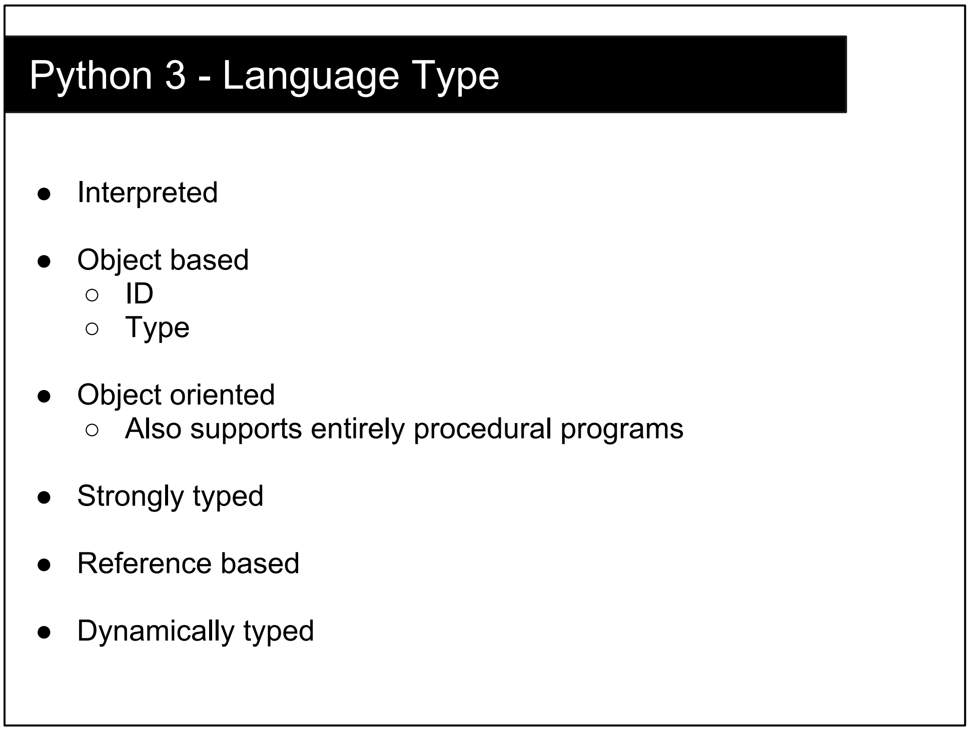
Python was created by Guido Van Rossum. He was also responsible for the initial implementations of the language. He continues to be actively involved in the development of the language and has been designated a BDFL (Benevolent Dictator For Life - having the final say in the language direction) by the Python community.

Python was initially modeled as a successor to the ABC language.

Python was first publicly released in 1991 onto a message board by Guido Van Rossum himself. At that point Guido had been working on Python for roughly two years.

The two current production quality versions of Python releases are (production) 2.7 and (testing) 3.3. Although the core for version 3 is different from version 2, some of the enhancements made to version 3 have also been backported and made available in 2.7 to allow for easier transition to 3.

At present, the copyright for Python is owned by the Python Software Foundation, a non-profit organization created to foster Python. Although throughout the years different organizations have been the copyright owners for Python.



## Python 3 - Language Type

- Interpreted
- Object based
    - ID
    - Type
- Object oriented
    - Also supports entirely procedural programs
- Strongly typed
- Reference based
- Dynamically typed

**Language Type**

Interpreted language. The program is first parsed into its' abstract syntax tree (AST), then compiled down into it's bytecode representation by the compiler. The bytecode is then executed by the interpreter on the host machine.

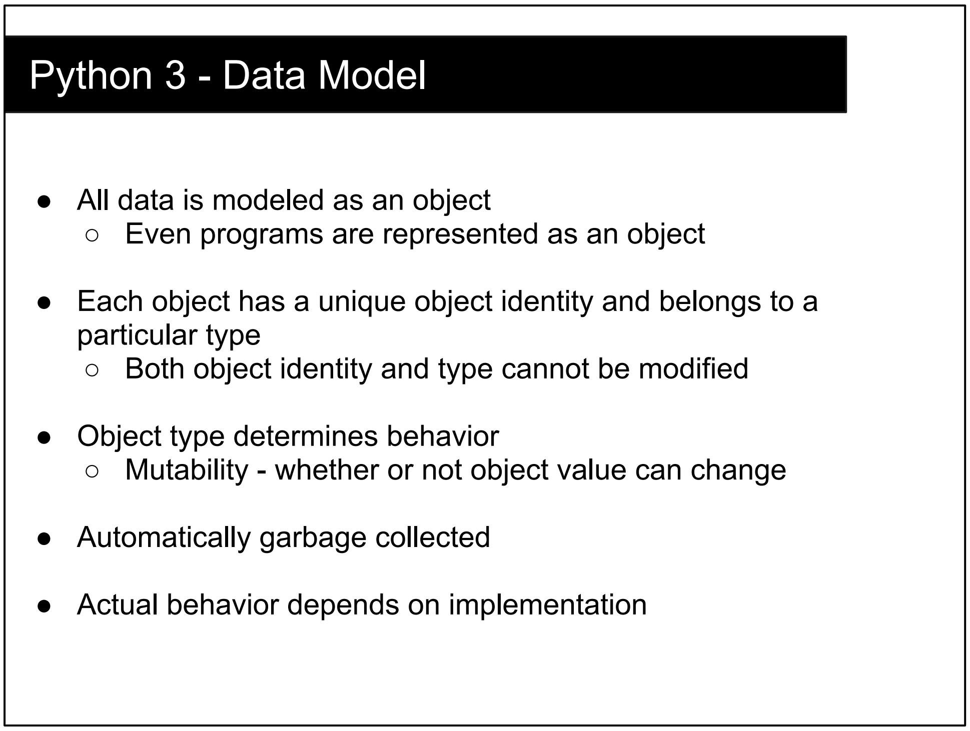
Object based. All data in python (even programs) are represented as objects. An object has a type and an id which remains with the object for it's lifetime. An object's type cannot be modified at any time after it's creation.

Object oriented. Simply means that Python also supports object oriented programming. It can also easily be used in a procedural manner. Python also supports entirely procedural programs.

Strongly typed. Every entity represented in a python program has a object (and hence a type) associated with it. The operations supported by an object is defined by the type it belongs to. (Simply stated Python does not permit operations between two different types. One such example would be concatenation of string and integer)

Reference based. Variables in python are simply reference type objects to the data it points to. References (variables) can at any time be modified to point to a different object.

Dynamically typed. The late association between an object and it's type renders it so that python can only support runtime type checking.



## Python 3 - Data Model

- All data is modeled as an object
  - Even programs are represented as an object
- Each object has a unique object identity and belongs to a particular type
  - Both object identity and type cannot be modified
- Object type determines behavior
  - Mutability - whether or not object value can change
- Automatically garbage collected
- Actual behavior depends on implementation

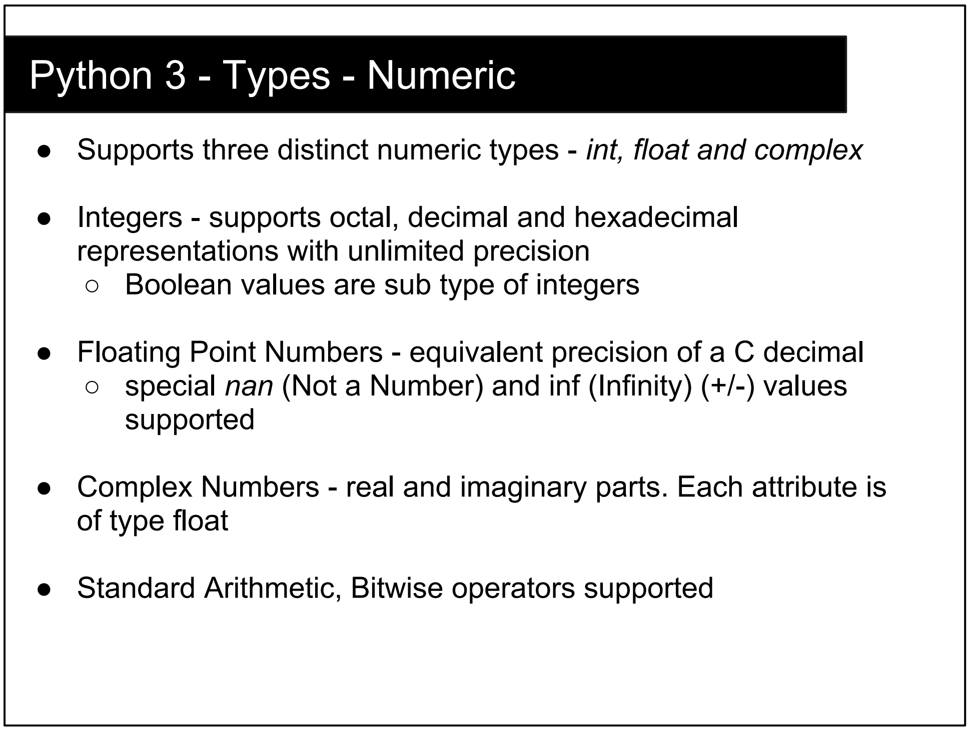
**Data Model**

In Python, all data is modeled as an object. Even a python program, which when parsed is also an object. (That infrastructure is made available as part of the Python standard library)

Regarding what an object contains. An object consists of a value and a type and an object identity. The type is the class associated with a particular object. Additionally, each object also has an object identity. This object identity is unique and along with the type is an unchangeable property for a given object.

The type becomes the core of the object, i.e. it determines the set of operations that the object will provide or will participate in. Along with the abstractions, the type also determines whether the value held by the object is mutable or not.

Finally, all entities in Python are automatically garbage collected. The actual semantics of how garbage collection is performed is a task left for the implementation to decide. Python also provides a module *'gc'* which allows a user to customize garbage collection to fit their needs.



## Python 3 - Types - Numeric

- Supports three distinct numeric types - *int, float and complex*
- Integers - supports octal, decimal and hexadecimal representations with unlimited precision
  - Boolean values are sub type of integers
- Floating Point Numbers - equivalent precision of a C decimal
  - special *nan* (Not a Number) and inf (Infinity) (+/-) values supported
- Complex Numbers - real and imaginary parts. Each attribute is of type float
- Standard Arithmetic, Bitwise operators supported

Now we will transition over to talk about built-in data types that you are most likely to encounter in Python

**Numeric**

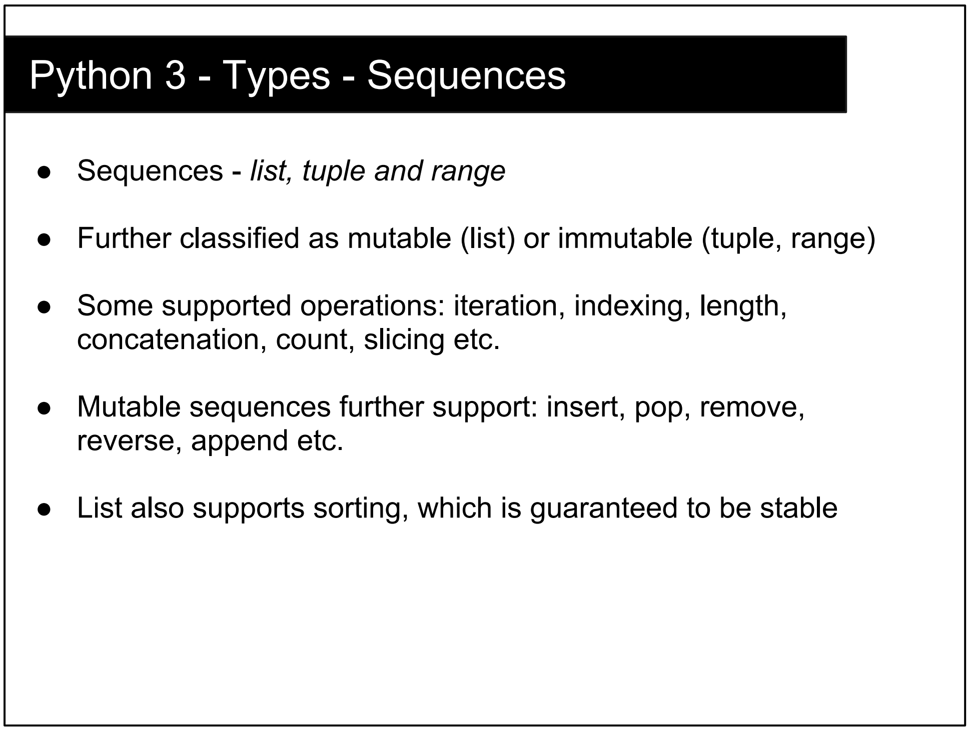
Python supports three basic numeric types namely integers, floating point numbers and complex numbers. All other numeric data types are specializations of these three types.

Integers: integers are the simplest of the three numeric data types. They contain a single value. The precision for which is unlimited. In Python integer types support decimal, octal and hexadecimal representations of a number.

Floating Point Numbers: This is the Python equivalent representation of a decimal type in C. There are two special values supported by floating point numbers: *nan* (Not a number) and *inf* (Infinity). Both these numbers are supported in both negative and positive axes.

Complex Numbers: This is Python representation of a complex number. It has a two parts (real and imag attributes) both of which are represented using a floating point number.

All three numeric types support arithmetic and bitwise operators. There are other additional operations that are supported as well.



## Python 3 - Types - Sequences

- Sequences - *list, tuple and range*
- Further classified as mutable (list) or immutable (tuple, range)
- Some supported operations: iteration, indexing, length, concatenation, count, slicing etc.
- Mutable sequences further support: insert, pop, remove, reverse, append etc.
- List also supports sorting, which is guaranteed to be stable

**Sequences**

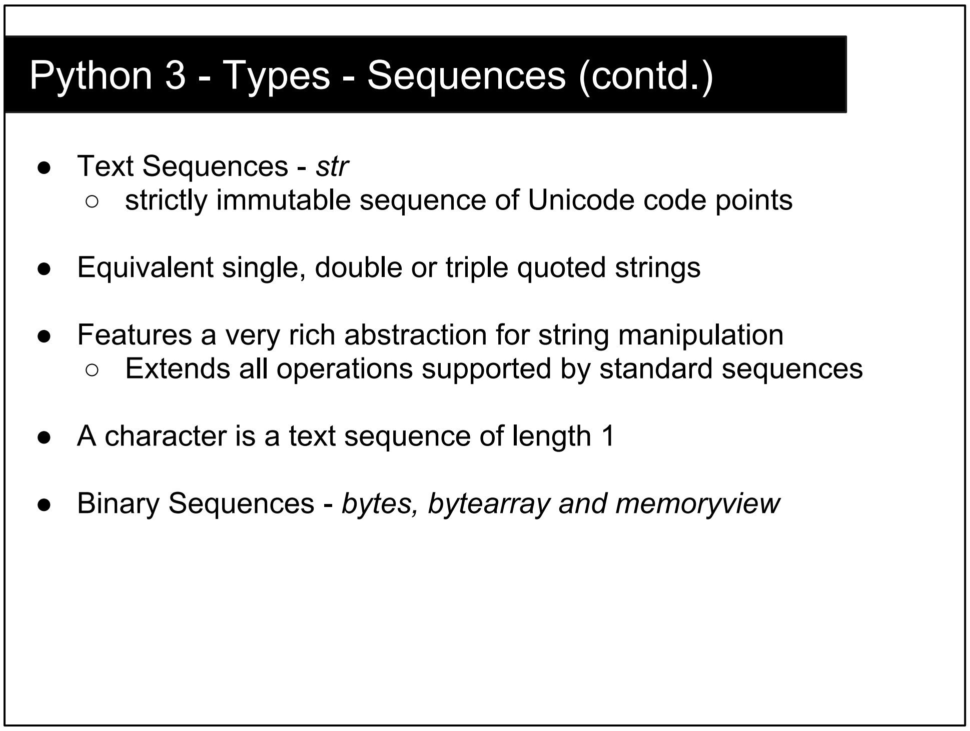
Sequences and Mappings in my opinion form the solid core which has permitted Python to be adopted in diverse areas of computing.

There are several different types of sequences. We will first discuss list, tuple and range sequences

You can further classify sequences by the type of the value they contain. Some sequences permit the value stored to be modified while others don't. These types of sequences which permit the value to be modified are known as Mutable sequences (list). The ones that do not permit values to be modified once set are known as Immutable (tuple, range)

As with any area in Python, the abstractions available in sequences are very rich.

List sequence type also supports sorting of the values contained. The actual semantics of the sort is left to the implementation, however Python promises that the sorting algorithm is stable. i.e. values that are equal in the sequence will maintain their original order



## Python 3 - Types - Sequences (contd.)

- Text Sequences - *str*
  - strictly immutable sequence of Unicode code points
- Equivalent single, double or triple quoted strings
- Features a very rich abstraction for string manipulation
  - Extends all operations supported by standard sequences
- A character is a text sequence of length 1
- Binary Sequences - *bytes, bytearray and memoryview*

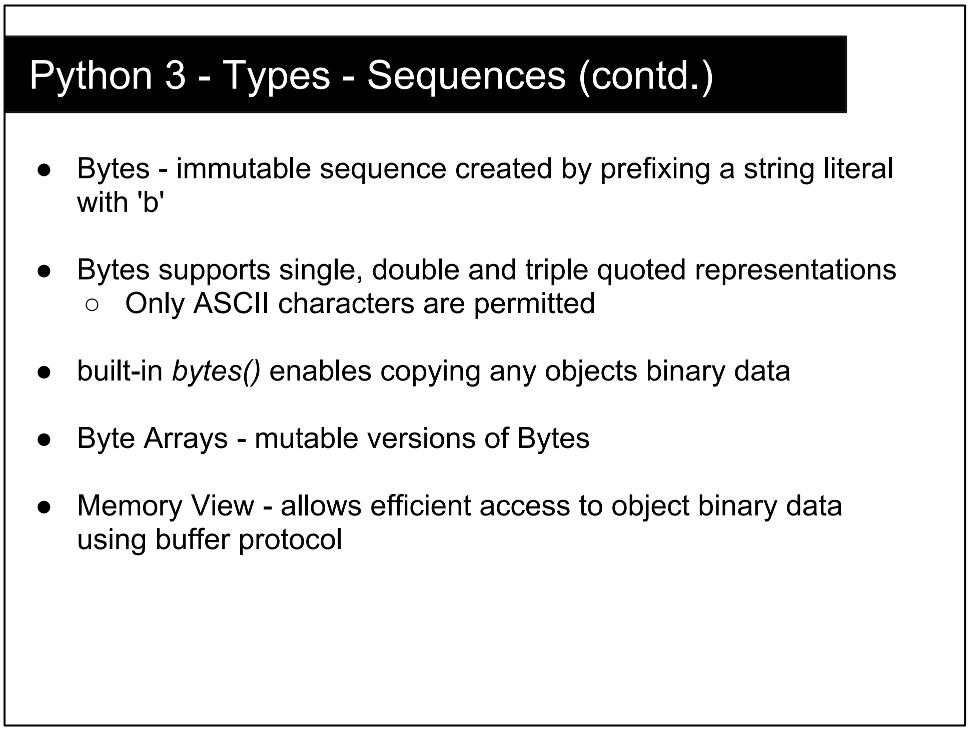
**Sequences (contd...)**

Text Sequences are immutable sequences of Unicode code points. This is a major enhancement added to Python 3 where the default string representation is in Unicode. Text sequences in python are called *string*

Text sequences can be constructed in three distinct ways: Single Quotes, Duoble Quotes and Triple Quoted string.
The language does not distinguish between the three different types aside from the range of possible valid characters. *string* class probably boasts the most extensive set of abstractions of any module.

Unlike languages of the early years, now a character is a text sequence which happens to have a length of one.

To wrap up sequences we will briefly look at the three different types of binary sequences supported in Python.
Those are *bytes, bytearray and memoryview*



## Python 3 - Types - Sequences (contd.)

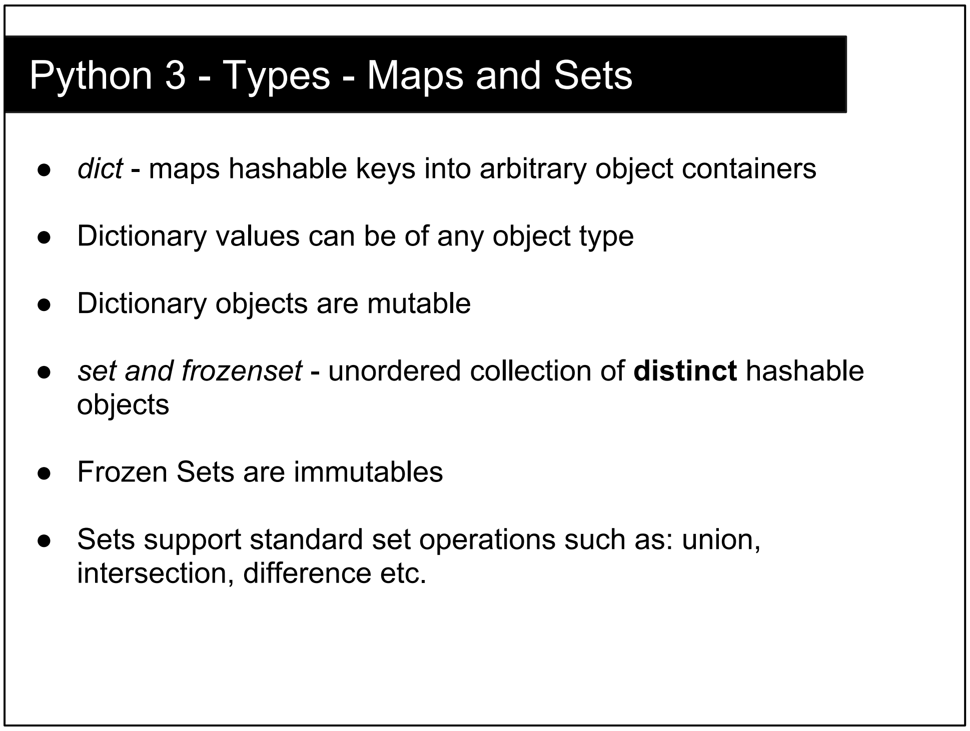
- Bytes - immutable sequence created by prefixing a string literal with 'b'
- Bytes supports single, double and triple quoted representations
  - Only ASCII characters are permitted
- built-in *bytes()* enables copying any objects binary data
- Byte Arrays - mutable versions of Bytes
- Memory View - allows efficient access to object binary data using buffer protocol

**Sequence (contd..)**

Bytes are an immutable sequences of single byte data. Bytes can can be created by prefixing a string sequence with a bytes prefix. There are numerous valid ways to represent a bytes prefix. This is the simplest way of representing binary data in Python. Since bytes only permit storing single byte data, any data set containing multi-byte code points must first be converted to an equivalent single byte (ASCII) representation.

Byte Arrays are the mutable version of the bytes. Byte arrays are created by calling their constructor - *bytearray*()

Memory View is a new addition to Python 3 which permits very efficient access to binary object data using buffer protocol.



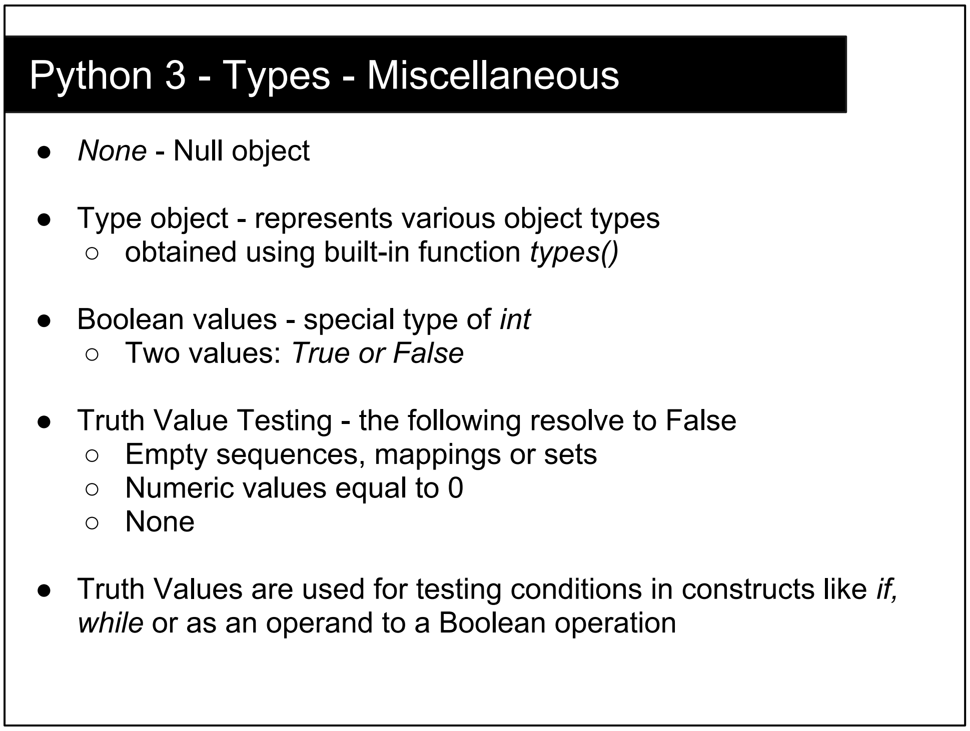
## Python 3 - Types - Maps and Sets

- *dict* - maps hashable keys into arbitrary object containers
- Dictionary values can be of any object type
- Dictionary objects are mutable
- *set and frozenset* - unordered collection of **distinct** hashable objects
- Frozen Sets are immutables
- Sets support standard set operations such as: union, intersection, difference etc.

**Maps and Sets**

Dictionary is the only built-in of type maps available. With a map, we hash keys into arbitrary object containers. Dictionary keys are required to be of type *string*. There is no restriction on the value referenced by each key. Also, dictionary objects are mutable. Assigning value by referencing a key will update the value stored in the dictionaries internal representation.

Sets are represented by *set* and *frozenset* data types. To distinguish sets from dictionaries; sets only permit storing an *unordered collection of distinct objects* that are hashable. A set is mutable while a frozenset is not. Sets honors most of the standard set operations like Union, Intersection etc.



## Python 3 - Types - Miscellaneous

- *None* - Null object
- Type object - represents various object types
  - obtained using built-in function *types()*
- Boolean values - special type of *int*
  - Two values: *True or False*
- Truth Value Testing - the following resolve to False
  - Empty sequences, mappings or sets
  - Numeric values equal to 0
  - None
- Truth Values are used for testing conditions in constructs like *if, while* or as an operand to a Boolean operation

**Remaining Types**

There are several other data types that are worth mentioning.

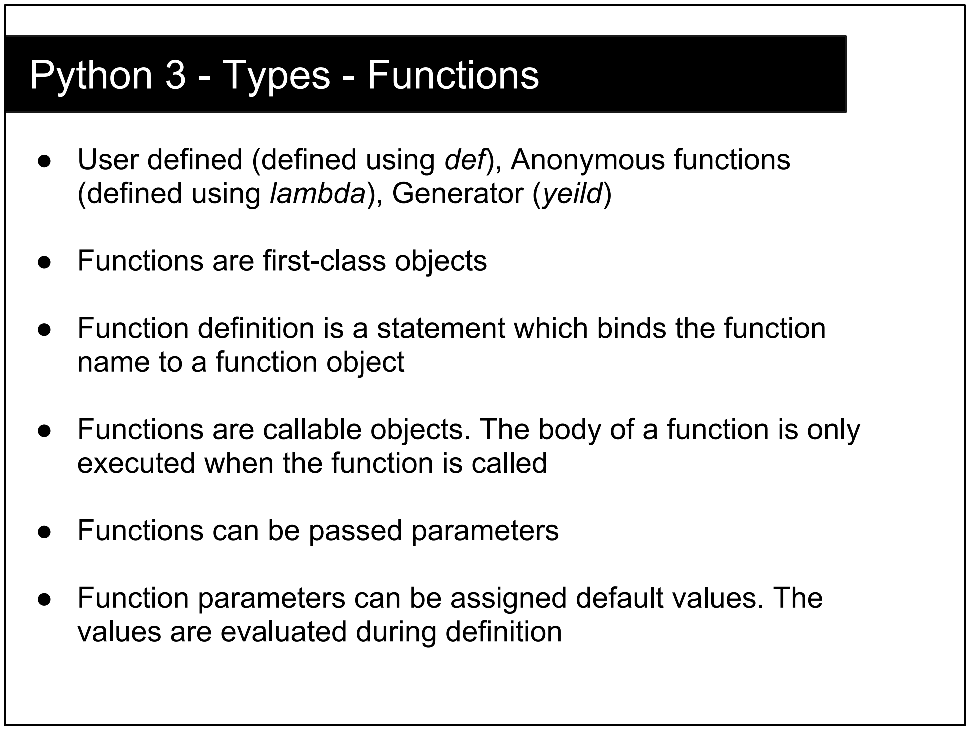
None - represents a null object. Signifies a lack of value.

Type object - represents the various object types available in the system. This object is obtained by using the built-in function types()

Boolean values - Firstly, Boolean values are a sub-type of the Integer type value. The only two valid values are True and False

**Truth Value Testing**

This is a way to identify what resolves to a True or False. This kind of testing is most commonly used in conditional constructs. Empty values or Zero resolves to a False. Everything else is True.



## Python 3 - Types - Functions

- User defined (defined using *def*), Anonymous functions (defined using *lambda*), Generator (*yeild*)
- Functions are first-class objects
- Function definition is a statement which binds the function name to a function object
- Functions are callable objects. The body of a function is only executed when the function is called
- Functions can be passed parameters
- Function parameters can be assigned default values. The values are evaluated during definition

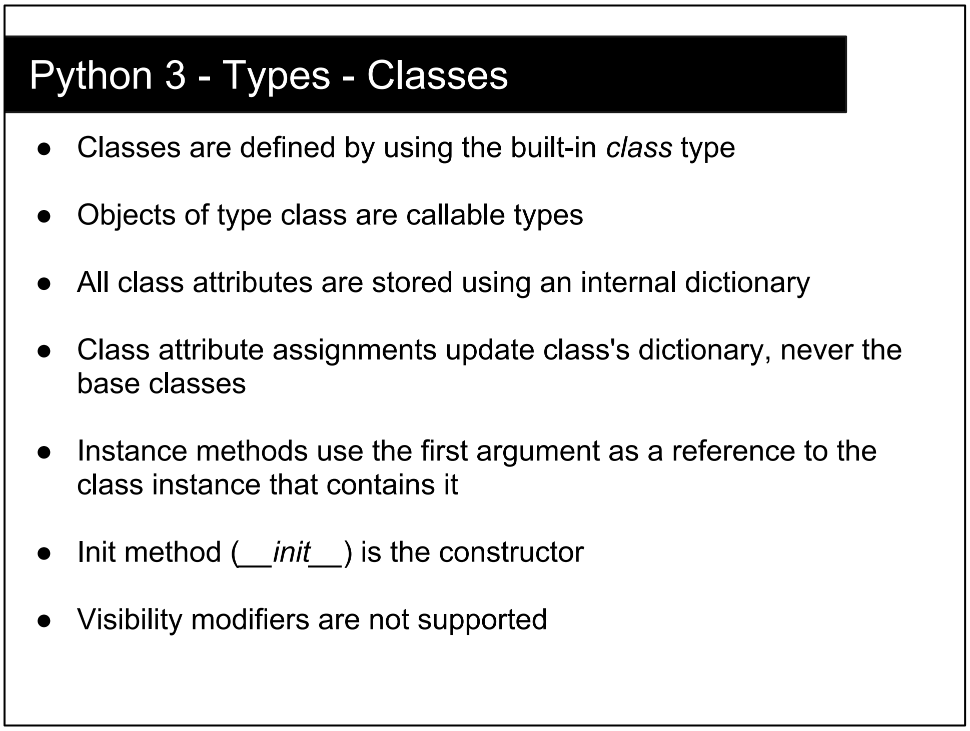
**Functions**

In Python, even functions are objects of a particular type. The broad group the functions belong to is known as the callable object types. There are several different types of objects that fall under the broad group of callable object types, here are a few:
1. User defined functions and Instance methods (defined using *def* keyword)
2. Generator expressions - (*yield* keyword)
2. Anonymous functions (defined using *lambda* expressions)

Furthermore, functions in python are first-class objects. I understood it to simply mean that function objects can be returned from and passed into a function. However, there is a good bit of detail related to scoping rules which will require a good exercise to digest.

Even function definition is a statement, which when the interpreter encounters is tasked with creating an instance of the function object and associating the name of the function with this object in the current local scope namespace. Also, as explained earlier functions belong to a general class of objects called callable object. When a function defined is called by using the '*()*' appended to the function name the expressions (code) inside the body of the function object is executed as a block.

We can always pass parameters to the functions (those that are defined to contain it). Also, function parameters can be assigned default values which are **evaluated at the time the function definition** is executed and not when the function is called.



## Python 3 - Types - Classes

- Classes are defined by using the built-in *class* type
- Objects of type class are callable types
- All class attributes are stored using an internal dictionary
- Class attribute assignments update class's dictionary, never the base classes
- Instance methods use the first argument as a reference to the class instance that contains it
- Init method (__*init*__) is the constructor
- Visibility modifiers are not supported

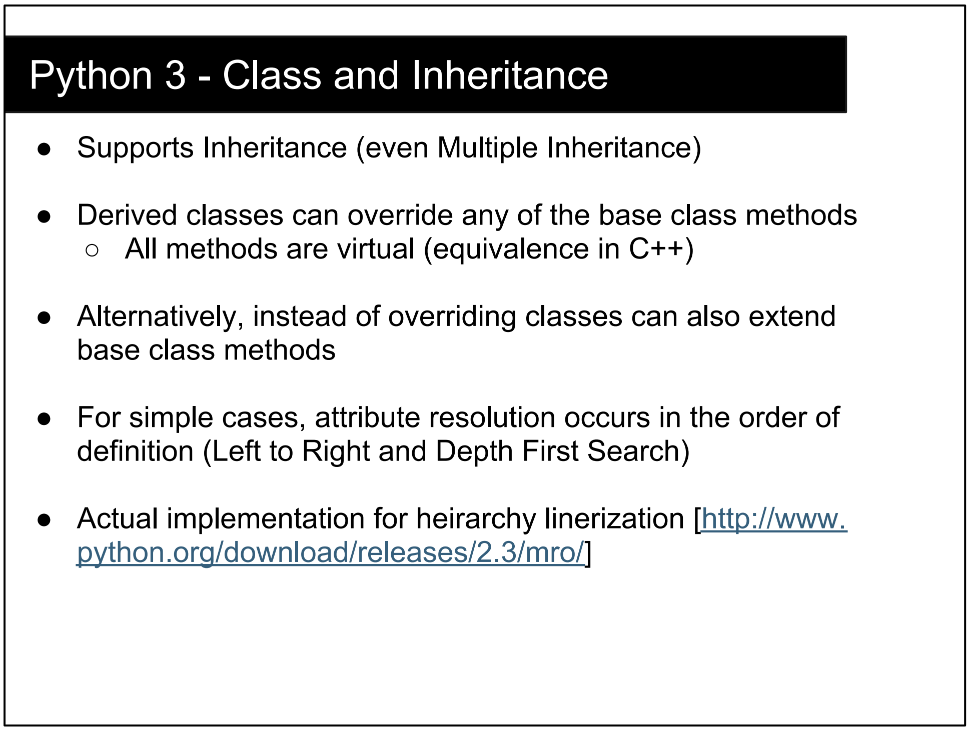
**Classes**

Similar to function objects class objects also belong to the group of callable objects. A class is defined by using the keyword *class*. When the class is defined, the name of the class is bound to the local namespace with the class object.

Every class attribute is stored using an internal class dictionary. When updating a class attributes value, the value associated with the key in the internal dictionary of the class is updated and never of the base classes of the class.

If a function needs to access instance attributes, the first argument passed to the method is a reference to the instance object itself. __init__ serves as the constructor

Python does not support visibility modifiers. i.e. aside from convention (using underscores in names) there is no way in python to hide either attributes or methods from being accessed from outside of the class itself. This is a significant difference when compared with OO programming languages like Java and C#.



## Python 3 - Class and Inheritance

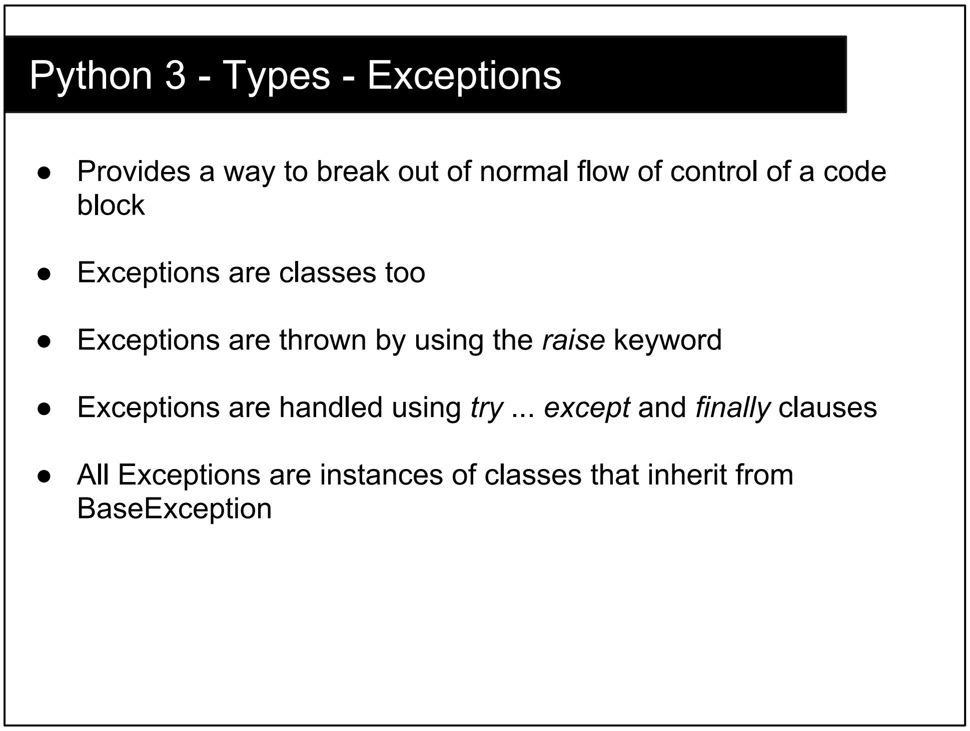
- Supports Inheritance (even Multiple Inheritance)
- Derived classes can override any of the base class methods
  - All methods are virtual (equivalence in C++)
- Alternatively, instead of overriding classes can also extend base class methods
- For simple cases, attribute resolution occurs in the order of definition (Left to Right and Depth First Search)
- Actual implementation for heirarchy linerization [http://www.python.org/download/releases/2.3/mro/]

**Class and Inheritance**

Python supports inheritance (surprise!). However, what is different from the more popular OO languages today (C#, Java) is that Python supports multiple inheritance.

Any of the derived classes can override any of the methods defined in any of the base classes. Effectively, every instance method defined is the equivalent of a C++ virtual function. Even when overriding methods, it it possible to access the base classes methods allowing us to also extend the behavior defined in base class methods.

With regards to inheritance, for the most simple of cases - attribute (and method) resolution occurs in the order of definition in the class definition object. This happens from Left to Right and uses Depth First Search. The actual implementation however is significantly more complex than a simple DFS. (Ahh well...)



## Python 3 - Types - Exceptions

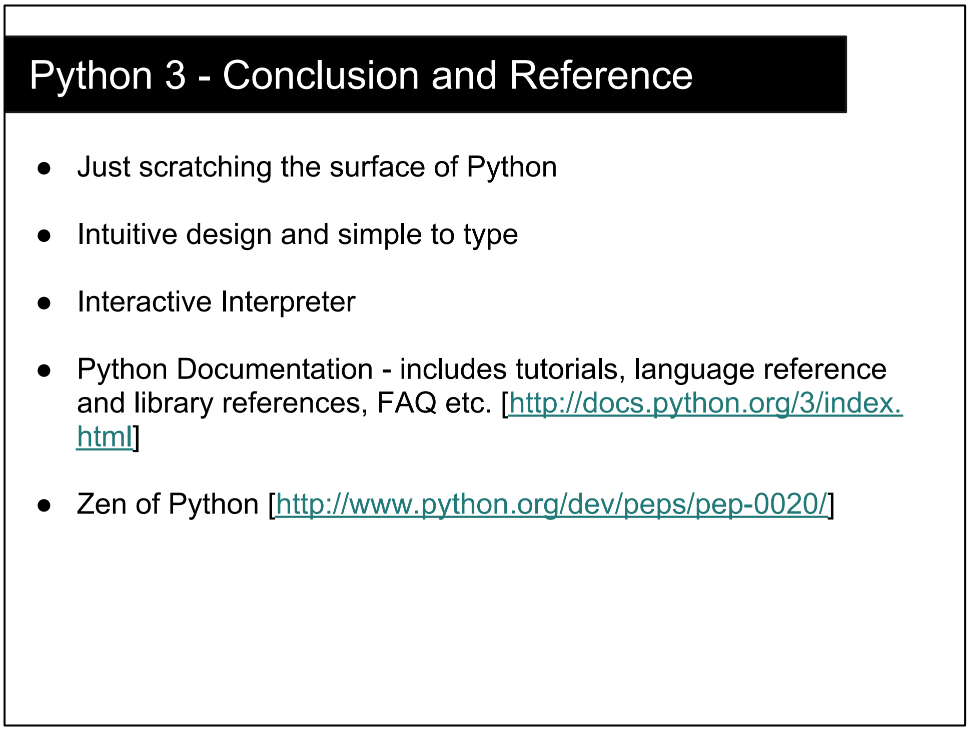
- Provides a way to break out of normal flow of control of a code block
- Exceptions are classes too
- Exceptions are thrown by using the *raise* keyword
- Exceptions are handled using *try* ... *except* and *finally* clauses
- All Exceptions are instances of classes that inherit from BaseException

**Exceptions**

Exceptions provide a way for the interpreter to break out of the normal flow of control for a code block. Exceptions also belong to the callable type and are special type of class objects. All exceptions inherit from a single built-in exception class called the BaseException

An exception in Python is thrown by using the *raise* keyword. If an exception is not specified, Python tries to locate the last exception thrown and raises that instead. The full set of constructs in Python to accomplish Exception handling are: *try ... except, finally and raise*

The method of exception handling that Python uses is known as "termination" model. This model makes it possible to handle different types of errors encountered however it is not possible to return to the source of the error and fix the problem.



## Python 3 - Conclusion and Reference

- Just scratching the surface of Python
- Intuitive design and simple to type
- Interactive Interpreter
- Python Documentation - includes tutorials, language reference and library references, FAQ etc. [http://docs.python.org/3/index.html]
- Zen of Python [http://www.python.org/dev/peps/pep-0020/]

**Conclusion**

Python is a very intuitive and a simple language to program in. The abstractions it provides seem like an extension of the english language.

Python is a language rich in it's definition and in the standard libraries that decorate it. We only explored some of the aspects of the type system provided in Python. It's this base that is extended in an simple manner to build this powerful language.

I encourage those of us that are new to Python to try and experiment with the language. Python provides a command line interpreter as part of the standard installation, this is a very helpful tool to understand the basics of the language and test code blocks out.

Thanks.