

# Introduction to Python

# What is Python?

- A programming language with strong similarities to PERL, but with powerful typing and object oriented features.
  - Useful built-in types (lists, dictionaries).
  - Clean syntax, powerful extensions.
  - Strong numeric processing capabilities: matrix operations, etc.
  - Free

# Why Python?

- High-level language, can do a lot with relatively little code
- Supposedly easier to learn than its main competitor, Perl
- Fairly popular among high-level languages
- Robust support for object-oriented programming
- Support for integration with other languages

# For More Information?

*Python Homepage:* <http://python.org/>

- documentation, tutorials, beginners guide, core distribution, ...

Books include:

- *Learning Python* by Mark Lutz
- *Python Essential Reference* by David Beazley
- *Python Cookbook*, by Martelli, Ravenscroft and Ascher  
(online at <http://code.activestate.com/recipes/langs/python/>)
- <http://wiki.python.org/moin/PythonBooks>

# Development Environments

1. IDLE ( we use this in the course)
2. PyDev with Eclipse
3. Emacs
4. Vim
5. Notepad++ (Windows)

Our platform: Python 3.x + IDLE + Windows 7

# Sample Code...

```
>>>x = 34 - 23                # A comment.
>>>y = "Hello"                # Another one.
>>>z = 3.45
>>>if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World"
>>>print(x)
>>>print(y)
```

# Enough to Understand

- Assignment uses = and comparison uses ==.
- For numbers +-\*/% are as expected.
- Logical operators are words (**and**, **or**, **not**) *not symbols* (&&, ||, !).
- The basic printing command is “print.”
- First assignment to a variable will create it.

# Look at a sample of code...

```
>>>x = 34 - 23                                     # A comment.
>>>y = "Hello"                                     # Another one.
>>>z = 3.45
>>>if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World"
>>>print(x)
>>>print(y)
```

indentation are important. No braces { } to mark blocks of code in Python. Use consistent indentation instead. The first line with a new indentation is considered outside of the block.



# Look at a sample of code...

Start comments with # – the rest of line is ignored.

```
>>>x = 34 - 23
```

```
>>>y = "Hello"
```

```
>>>z = 3.45
```

```
>>>if z == 3.45 or y == "Hello":
```

```
    x = x + 1
```

```
    y = y + " World"
```

```
>>>print x
```

```
>>>print y
```

```
# A comment.
```

```
# Another one.
```

indentation are important. No braces { } to mark blocks of code in Python. Use consistent indentation instead. The first line with a new indentation is considered outside of the block.

# Variable types

- Dynamic type :
  - Variable types don't need to be declared.
  - Python figures out the variable types on its own.
  - “variables” in python: *object references*, similar to C/C++ pointers.
- Strong Type
  - Python restrictive about how types can be intermingled
    - Try `x+y`, what happened? Error

# Variable types

- Examples:

```
>>> pi = 3.1415926
```

```
>>> message = "Hello, world"
```

```
>>> i = 2+2
```

```
>>> type(pi)
```

```
>>> type(message)
```

```
>>> type(i)
```

Output:

```
<type 'float'>
```

```
<type 'str'>
```

```
<type 'int'>
```

# Variable names

- Can contain letters, numbers, and underscores
- Must begin with a letter
- There are some reserved words:

and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while

# Operators

- + addition
- - subtraction
- / division
- \*\* exponentiation
- % modulus (remainder after division)
- +=, -=, /= .... (No ++ or --)

# Operators

- Examples

```
>>> 2*2
>>> 2**3
>>> 10%3
>>> 1.0/2.0
>>> 1/2
```

Output:

```
4
8
1
0.5
0
```

# Type conversion

- `int()`, `float()`, `str()`, and `bool()`
- |   |                  |
|---|------------------|
| <code>&gt;&gt;&gt; 1.0/2.0</code>           | • Output:<br>0.5 |
| <code>&gt;&gt;&gt; 1/2</code>               | 0                |
| <code>&gt;&gt;&gt; float(1)/float(2)</code> | 0.5              |
| <code>&gt;&gt;&gt; int(3.1415926)</code>    | 3                |
| <code>&gt;&gt;&gt; str(3.1415926)</code>    | 3.1415926        |
| <code>&gt;&gt;&gt; bool(1)</code>           | True             |
| <code>&gt;&gt;&gt; bool(0)</code>           | False            |

# Operators acting on strings

- `>>> "NUS!"*3`  
`'NUS!NUS!NUS!'`
- `>>> "hello " + "world!"`  
`'hello world!'`



# Input from keyboard

- `number = input("Enter Your Lucky Number: ")`  
`print(number)`

```
name = raw_input("Enter Your loved Pets: ")  
print(name)
```

use `help(input)` or `help(raw_input)` to get help from IDLE

# Conditionals

- True and False booleans
- Comparison and Logical Operators
- if, elif, and else statements

# Booleans: True and False

## Things that are False

- The boolean value False
- The numbers 0 (integer), 0.0 (float) and 0j (complex).
- The empty string "".
- The empty list [], empty dictionary {} and empty set set().

## Things that are True

- The boolean value True
- All non-zero numbers.
- Any string containing at least one character.
- A non-empty data structure.

boolean expression: evaluated as True or False

# Comparison operators

- `==` : is equal to?
- `!=` : not equal to
- `>` : greater than
- `<` : less than
- `>=` : greater than or equal to
- `<=` : less than or equal to
- `is` : do two references refer to the same object?

- Can “chain” comparisons:

```
>>> a = 49
```

```
>>> 0 <= a <= 99
```

```
True
```

# Logical operators

- and, or, not

```
>>> 2+2==5 or 1+1==2
```

```
True
```

```
>>> 2+2==5 and 1+1==2
```

```
False
```

```
>>> not(2+2==5) and 1+1==2
```

```
True
```

- **We do NOT use &&, ||, !**

# If Statement

```
>>> ABC = "NUS"
```

```
>>> if not ABC: Must have ":" here  
    print("The ABC string is empty")
```

- The “else” case is always optional

# Elif Statement

- Equivalent of “else if” in C

- Example:

```
x = 3
if (x == 1):
    print("one")
elif (x == 2):
    print("two")
else:
    print("many")
```

No switch in python

# Iteration

- while loops
- for loops
- flow control within loops: break, continue
- range function



# While Statement

```
>>> Schools = ["soc", "fass", "eng"]
>>> i = 0
>>> while i<len(Schools)
        print(Schools[i])
        i = i + 1
```

Output:

**soc**

**fass**

**eng**

# For Statement

```
>>> Schools = ["soc", "fass", "eng"]
>>> for name in Schools:
    print(name)
>>> for i in range(4):
    print(i)
```

# Break, continue

```
>>> for value in [3, 1, 4, 1, 5, 9, 2]:  
    print("Checking", value)  
    if value > 8:  
        print("Exiting for loop")  
        break  
    elif value < 3:  
        print("Ignoring")  
        continue  
    print("The square is", value**2)
```

Use "break" to stop  
the for loop

Use "continue" to stop  
processing the current item

# Range()

- creates a list of numbers in a specified range
- `range([start,] stop[, step])` -> list of integers

```
>>> range(5)
```

```
[0, 1, 2, 3, 4]
```

```
>>> range(5, 10)
```

```
[5, 6, 7, 8, 9]
```

```
>>> range(0, 10, 2)
```

```
[0, 2, 4, 6, 8]
```

# Datatypes

- Numbers
  - Integer
  - Float
- String
- Collection Data Types: List/Tuple/Dictionary

# String basics

- Strings can be delimited by single or double quotes. Multi-line strings use triple-quotes
- An empty string is denoted by having nothing between string delimiters (e.g., "")
- Can access elements of strings with [], with indexing starting from zero:  

```
>>> "snakes"[3]  
'k'
```

Note: can't go other way --- can't set "snakes"[3] = 'p' to change a string; strings are *immutable*

- a[-1] gets the *last* element of string a (negative indices work through the string backwards from the end)

# String basics

```
>>> ABC = "hello,NUS"
```

```
>>> ABC[0]
```

```
'h'
```

```
>>> ABC[1]
```

```
'e'
```

```
>>> ABC[-1]
```

```
's'
```


```
>>> ABC[0:5]
```

```
'hello'
```

```
>>> ABC[1:-1]
```

```
'ello,NU'
```

Use "slice" notation to  
get a substring



ABC[start:end:step]

# String basics

Type conversion:

```
>>> int("42")
```

```
42
```

```
>>> str(20.4)
```

```
'20.4'
```

Compare strings with the is-equal operator

```
>>> a = "hello"
```

```
>>> b = "hello"
```

```
>>> a == b
```

```
True
```

- ```
>>location = "Chattanooga " + "Tennessee"
```

```
>>>Chattanooga Tennessee
```



# String formatting

- ```
>>> greeting = "Hello"
>>> "%s. Welcome to python." % greeting
'Hello. Welcome to python.'
```
- ```
>>> "The grade for %s is %4.1f" % ("Tom",
76.051)
'The grade for Tom is 76.1'
```

# String methods

- built-in methods for string
- S.capitalize()
- S.center(width)
- S.count(substring [, start-idx [, end-idx]])
- S.find(substring [, start [, end]])
- S.isalpha(), S.isdigit(), S.islower(), S.isspace(), S.isupper()
- S.join(sequence)
- And many more!

# find, split

```
smiles = "C(=N)(N)N.C(=O)(O)O"
```

```
>>> smiles.find("(O)")
```

```
15
```

Use “find” to find the start of a substring.

```
>>> smiles.find(".")
```

```
9
```

Start looking at position 10.

```
>>> smiles.find(".", 10)
```

```
-1
```

Find returns -1 if it couldn't find a match.

```
>>> smiles.split(".")
```

```
['C(=N)(N)N', 'C(=O)(O)O']
```

Split the string into parts with “.” as the delimiter

# replace

- Makes a new string with the replacement performed:

```
>>> a = "abcdefg"
```

```
>>> b = a.replace('c', 'C')
```

```
>>> b
```

```
abCdefg
```

```
>>> a
```

```
abcdefg
```

# in, not in

```
>>> if "Br" in "Brother":  
    print("contains brother")
```

```
>>> email_address = "clin"
```

```
>>> if "@" not in email_address:  
    email_address += "@brandeis.edu"
```

# Datatypes

- Numbers
  - Integer
  - Float
- String
- Collection Data Types: List/Tuple/Dictionary

# Tuples

- A collection of data items which may be of different types.
- Tuples are *immutable*

```
>>>"Tony", "Pat", "Stewart"  
( 'Tony', 'Pat', 'Stewart' )
```

- Python uses *()* to denote tuples
- An empty tuple is denoted by *()*
- Can be used to return multiple values in function call

# Tuples

```
>>> yellow = (255, 255, 0) # r, g, b
```

```
>>> yellow[1:]
```

```
(255, 0)
```

```
>>> yellow[0] = 0
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

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



# Lists

- Like tuples, but *mutable*, and designated by **square brackets []** instead of parentheses:

```
>>> [1, 3, 5, 7, 11]
```

```
[1, 3, 5, 7, 11]
```

```
>>> [0, 1, 'boom']
```

```
[0, 1, 'boom']
```

- An empty list is []

# List Indexing

```
names = ["Ben", "Chen", "Yaqin"]
```

```
>>> names[0]
```

```
'Ben'
```

```
>>> names[3] (error)
```

Out of range values  
raise an exception

```
>>> names[-1]
```

```
'Yaqin'
```

Tuple also uses “[]” for elements indexing. In list/tuple, they contains *object references*

```
>>> names = ["Ben", "Chen", (1, 2, 3)]
```

```
>>> print(names[2][1])
```

```
2
```

# List Methods

Use `len()` to get the length of a list

```
>>> names = ["Ben", "Chen", "Yaqin"]  
>>> len(names)  
3
```

Append an new item in list

```
>>> x = [1, 2, 3]  
>>> x.append("done")  
>>> print(x)  
[1, 2, 3, 'done']
```

# List Methods

```
>>> ids = ['hello', '2plv', '1crn', '1alm']
>>> del ids[0]
>>> ids
['2plv', '1crn', '1alm']
>>> ids.sort()
>>> ids
['1alm', '1crn', '2plv']
>>> ids.reverse()
>>> ids
['2plv', '1crn', '1alm']
>>> ids.insert(0, "9pti")
>>> ids
['9pti', '2plv', '1crn', '1alm']
```

# Dictionaries

- Unordered collections where items are accessed by a key, not by the position in the list
- Collection of arbitrary objects; use object references like lists
- Nestable
- Can grow and shrink in place like lists
- Concatenation, slicing, and other operations that depend on the order of elements do not work on dictionaries

# Dictionary

- They map from a “key” to a “value”.

```
symbol_to_name = {  
    'David': 'Professor',  
    'Sahan': 'Postdoc',  
    'Shawn': 'Grad student'  
}
```

- Duplicate keys are not allowed
- Duplicate values are just fine
- Keys must be immutable

# Dictionary

- Change in place(*Mutable*)

```
>>> jobs['Shawn'] = 'Postdoc'
>>> jobs['Shawn']
'Postdoc'
```

- Lists of keys and values

```
>>> jobs.keys()
['Sahan', 'Shawn', 'David'] # note order is
diff
>>> jobs.values()
['Postdoc', 'Postdoc', 'Professor']
>>> jobs.items()
[('Sahan', 'Postdoc'), ('Shawn',
'Postdoc'), ('David', 'Professor')]
```

# Dictionary

- Delete an entry (by key)  
`del d['keyname']`
- Add an entry  
`d['newkey'] = newvalue`
- See if a key is in dictionary  
`d.has_key('keyname')` or `'keyname' in d`
- `get()` method useful to return value but not fail (return `None`) if key doesn't exist (or can provide a default value)  
`d.get('keyval', default)`



# Dictionary

- Going through a dictionary by keys:

```
>>>bookauthors = {'Gone': 'Margaret  
Mitchell',  
                  'Aeneid': 'Virgil',  
                  'Odyssey': 'Homer'}  
>>>for book in bookauthors:  
    print(book, 'by', bookauthors[book])
```

output:

Gone by Margaret Mitchell

Aeneid by Virgil

Odyssey by Homer

# Constructing dictionaries from lists

- If we have separate lists for the keys and values, we can combine them into a dictionary using the zip function and a dict constructor:

```
>>>keys = ['david', 'chris', 'stewart']  
>>>values = ['504', '637', '921']  
>>>D = dict(zip(keys, vals))
```

# Functions

- Defining functions
- Scope
- Return values

# Functions

- Define them in the file above the point they're used

```
>>>def square(n):  
    return n*n
```

```
>>>print("The square of 3 is ", square(3))
```

Output:

```
The square of 3 is 9
```

# More about functions

- Arguments are optional. Multiple arguments are separated by commas.
- If there's no return statement, then “None” is returned. Return values can be simple types or tuples. Return values may be ignored by the caller.
- Functions are “typeless” Can call with arguments of any type, so long as the operations in the function can be applied to the arguments.

# Scope

- Variables declared in a function do not exist outside that function
- ```
>>>def square(n):  
    m = n*n  
    return m
```

```
print(m)
```

Output:

```
NameError: name 'm' is not defined
```

# Scope

- Variables assigned within a function are local to that function call
- Variables assigned at the top of a module are global to that module; there's only “global” within a module
- Within a function, first search the variable in the local scope, if fail then in global scope (must be declared as “global”)

# Scope example

```
>>>a = 5                # global
```

```
>>>def func(b):  
    c = a + b  
    return c
```

```
>>>func(4)              # gives 4+5=9
```

```
>>>c                    # not defined
```



# Scope example

```
>>>a = 5 # global
```

```
>>>def func(b):
    global c
    c = a + b
    return c
```

```
>>>func(4)      # gives 4+5=9
```

```
>>>c # now it's defined (9)
```

# By value / by reference

- Python acts like C's pass by pointer; in-place changes to mutable objects can affect the caller

# Example

```
>>>def f1(x,y):  
    x = x * 1  
    y = y * 2  
    print(x, y)
```

```
# 0 [1, 2, 1, 2]
```

```
>>>def f2(x,y):  
    x = x * 1  
    y[0] = y[0] * 2  
    print(x, y)
```

```
# 0 [2, 2]
```

```
>>>a = 0  
>>>b = [1,2]  
>>>f1(a,b)  
>>>print a, b  
>>>f2(a,b)  
>>>print(a, b)
```

```
# 0 [1, 2]
```

```
# 0 [2, 2] b is changed
```

# Multiple return values

- Can return multiple values as follows:

```
>>>def onetwothree(x):  
    return x*1, x*2, x*3
```

```
>>>a,b,c = onetwothree(3)
```

```
>>>3, 6, 9
```

# Default arguments

- can define a function to supply a default value for an argument if one isn't specified

```
>>>def print_error(lineno,  
message="error"):  
>>>    print("%s at line %d" % (message,  
lineno))
```

```
>>>print_error(42)  
error at line 42
```

# Chapter 9: Python Modules

- Basics of modules
- Import and from ... import statements
- Changing data in modules
- Reloading modules
- Module packages
- Import as statement

# Module basics

- Each file in Python is considered a module. Everything within the file is encapsulated within a namespace (which is the name of the file)
- To access code in another module (file), import that file, and then access the functions or data of that module by prefixing with the name of the module, followed by a period
- Can import user-defined modules or some “standard” modules like `sys` and `random`

# Python standard library

- There are over 200 modules in the Standard Library:
  - eg: os sys math
- Consult the Python Library Reference Manual, included in the Python installation and/or available at <http://www.python.org>



# import the math module

```
>>> import math
```

```
>>> math.pi
```

```
3.1415926535897931
```

```
>>> math.cos(0)
```

```
1.0
```

```
>>> math.cos(math.pi)
```

```
-1.0
```

```
>>> dir(math)
```

```
['__doc__', '__file__', '__name__', '__package__', 'acos',  
    'acosh',  
    'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',  
    'cosh', 'degrees', 'e', 'exp', 'fabs', 'factorial', 'floor', 'fmod',  
    'frexp', 'fsum', 'hypot', 'isinf', 'isnan', 'ldexp', 'log', 'log10',  
    'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',  
    'tanh', 'trunc']
```

# “import” and “from ... import ...”

```
>>> import math
```

```
math.cos
```

```
>>> from math import cos, pi
```

```
cos    #there is no sin, as we only import cos and pi
```

```
>>> from math import *
```

**Start Programming !**