Overview

- Fast, dynamically-typed, and extensible scripting language
- Multi-paradigmed: Object-Oriented, Functional, Procedural, Imperative
- Commonly used for:
  - Scientific Computing
  - Automating tasks
  - Working with databases
  - Web development
  - Game development
  - Text-processing
  - AI
History

• Designed by Guido Van Rossum in early 90’s – Very young language
• Many features of Python were inspired by the interpreted language ABC
• Rossum wanted to fix some of ABC’s issues while retaining some of it’s useful features
  • Data manipulation
  • Simple and quickly increased programmer’s productivity
History (Evolution)

• Python Version 1.0
  • Functional programming tools such as lambda, map, filter, and reduce

• Python Version 2.0
  • List comprehensions (borrowed from Haskell), garbage collection system, expansion of numerous modules, bug-fixes for popular libraries/modules

• Python Version 3.0
  • Rectified some fundamental design flaws
    • Print statement -> Print function
    • Basic math operations
    • Overhaul for iterators
    • _future_ module (allowed some necessary backwards-compatibility to Version 2.7)

• Currently on Version 3.5
Language Concepts

• Interactive shell built-in
  • Allows for quick testing of code snippets in isolation

• Requires careful attention of line spacing and indentation
  • Python uses indents instead of braces for classes, functions, control-structures

• Multiple statements on a single line
  • Can make code much cleaner if used appropriately
  • Popular among experienced programmers, known as “one-liners”

• Generators
  • Lazy evaluation
  • Computations over potentially huge sets of data without compromising memory
Example of using a generator:

If we ran this test:

- The entire list (100,000 items long) is created only one value at a time.
- Say we were looking for the 1,000th iteration...
- The list version would calculate the entire sequence but the generator could stop at our desired target

```
# function version
def fibon(n):
    a = b = 1
    result = []
    for i in xrange(n):
        result.append(a)
        a, b = b, a + b
    return result
```

```
# generator version
def fibon(n):
    a = b = 1
    for i in xrange(n):
        yield a
        a, b = b, a + b
```
Language Concepts

• Python’s compatibility with other languages
  • Tools/libraries that allow Python to integrate with other major languages
    • C/C++ with Pyrex/Cython
    • Java with Jython/Jpype
    • C#/.NET with IronPython
    • Fortran with F2PY
    • Prolog with PyLog
    • And many, many more

• Everything in Python is an object
  • Can be assigned to a variable or passed as an argument
    • “foo = Foo( )”
    • “foo = 10”
    • “foo = “Hello world!”
  • This is where “dynamically-typed” comes into play, although Python is still strongly-typed
Language Concepts

- Python is very high-level making it easy to read/write compared to other languages
- Use of “for-else” control statements
- Swapping values of variables dynamically
- Argument unpacking
  - See these in example program
- Assigning multiple values to a variable

```
“Hello, World”

C
#include <stdio.h>
int main(int argc, char ** argv)
{
    printf("Hello, World!\n");
}

Java
public class Hello
{
    public static void main(String argv[])
    {
        System.out.println("Hello, World!");
    }
}

now in Python
print “Hello, World!”
```
Example

```python
# For-else block:
def newControls(thislist):
    print("Input list =", thislist)
    for element in thislist:
        if element == "target":
            print(element, "= target")
            print("Search successful!")
            break
        elif element != "target":
            print(element, "!= target")
    else:
        print("Your target is not in this list!")

# Assigning multiple values to a variable:
def multvals():
    print("Running 'temp = 1,2,3' assigns variable a tuple of three values")
    temp = 1,2,3
    print("Now, 'temp' =", temp)
    print("Running 'x,y,z = temp' dynamically assigns the values from temp to x,y, and z")
    x,y,z = temp
    print("Now, x =", x)
    print("Now, y =", y)
    print("Now, z =", z)

# Executing functions:
def splitThree(x,y,z):
    print(x)
    print(y)
    print(z)

def changeArgs(*args):
    args = list(args)
    args[0] = 'Hello'
    args[1] = 'awesome'
    splitThree(*args)

# Testing:
def testing():
    print("Testing for-else block:")
    newControls([24, "Python", [], 2])
    print("")
    newControls(["NOPE", .000001, "target"])
    print("")
    print("Testing argument unpacking:")
    changeArgs("Goodbye", "cruel", "world!")
    print("")
    print("Testing assigning multiple values to a variable:")
    multvals()

testing()
```
Example

Python 3.5.0 (v3.5.0:374f501f9567, Sep 13 2015, 02:16:59) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

>>> 
-------- RESTART: C:\Python3.5\Python Files\presentationExample.py --------

Testing for-else block:
Input list = [24, 'Python', []]
24 != target
Python != target
[] != target
Your target is not in this list!

Input list = ['NOPE', 1e-06, 'target']
NOPE != target
1e-06 != target
target = target
Search successful!

Testing argument unpacking:
Hello
awesome
world!

Testing assigning multiple values to a variable:
Running 'temp = 1,2,3' assigns variable a tuple of three values
Now, 'temp' = (1, 2, 3)
Running 'x,y,z = temp' dynamically assigns the values from temp to x,y, and z
Now, x = 1
Now, y = 2
Now, z = 3
>>> testString = "Testing Python's interactive shell"
>>> testString
"Testing Python's interactive shell"
>>> print(type(testString))
<class 'str'>
>>> testString == 3
False
>>> |
Comparsions - C++

• Both object-oriented, imperative languages
• C++ compiled code to hardware native code language, but Python compiled to bytecode, executed by VM
• Usually code length of Python is 5 - 10 times shorter than equivalent C++ code.
• Python shines as a glue language, used to combine components written in C++. 

11/24/2015
Comparisons - Java

• Usually code length is 3 - 5 times shorter than the equivalent Java code.

• Python can be used to prototype components into Java implementation.

• The components can be developed in Java and then use it in Python. It can combined to form applications in Python.