Refresher Introduction to Python**¶**

A 5-min whirlwind refresher to the language

What and why**¶**

* interpreted (not compiled)
* object-oriented
* high-level
* dynamic semantics (aka execution semantics)
* fully dynamic typing
* dynamic binding
* quick development
* simple, readable, easy to learn syntax
* general purpose
* low program manteinance cost
* modularity and code reuse
* no licensing costs
* extensive standard library, "batteries included"
* imperative and functional programming
* automatic memory management (garbage-collected)
* Around 25 years old
* Versioning:
  + 2 parallel version streams
  + 2.x is fully-backward compatible
  + 3.x fixes some design, performance flaws
  + ... but breaks backward-compatibility
* **We'll be using 2.7.10+ for the training**

Expressions**¶**

In [ ]:

3+ 5  
3 + (5\*4)  
3 \*\* 2  
'Hello' + 'World'

Variables**¶**

In [ ]:

a = 4 << 3  
b = a \* 4.5  
c = (a+b)/2.5  
a = "Hello World"

* Variables are dynamically typed (No explicit typing, types may change during execution).
* Variables are just names for an object. Not tied to a memory location like in C

Conditionals**¶**

In [ ]:

# Compute maximum (z) of a and b  
if a < b:  
 z = b  
else:  
 z = a  
   
# empty body-block  
if a < b:  
 pass # Do nothing  
else:   
 z= a

* Indentation used to denote bodies.
* pass used to denote an empty body.
* There is no ’?:’ operator.

elif**¶**

In [ ]:

if a == '+':  
 op = PLUS  
elif a == '-':  
 op = MINUS  
elif a == '\*':  
 op = MULTIPLY  
else:  
 op = UNKNOWN

There's no switch statement

Conditionals**¶**

In [ ]:

if b >= a and b <= c:  
 print "b is between a and c"  
if not (b < a or b > c):  
 print "b is still between a and c"

Numbers**¶**

In [ ]:

a = 3 # Integer  
b = 4.5 # Floating point  
c = 517288833333L # Long integer (arbitrary precision)  
d = 4 + 3j # Complex (imaginary) number

Strings**¶**

In [ ]:

a = 'Hello'  
b = "World"  
c = "Bob said ’hey there.’" # A mix of both  
d = '''A triple quoted string  
can span multiple lines  
like this'''  
e = """Also works for double quotes"""

Lists**¶**

In [ ]:

a = [2, 3, 4] # A list of integers  
b = [2, 7, 3.5, "Hello"] # A mixed list  
c = [] # An empty list  
d = [2, [a,b]] # A list containing a list  
e = a + b # Join two lists

List: acessing elements**¶**

In [ ]:

x = a[1] # Get 2nd element (0 is first)  
y = b[1:3] # Return a sublist  
z = d[1][0][2] # Nested lists  
b[0] = 42 # Change an element

Tuples**¶**

In [ ]:

f = (2,3,4,5) # A tuple of integers  
g = (,) # An empty tuple  
h = (1,) # A single-element tuple  
h = (2, [3,4], (10,11,12)) # A tuple containing mixed objects

Tuple: Accessing elements**¶**

In [ ]:

x = f[1] # Element access. x = 3  
y = f[1:3] # Slices. y = (3,4)  
z = h[1][1] # Nesting. z = 4

Dict**¶**

In [ ]:

a= { } # An empty dictionary  
b = { 'x': 3, 'y': 4 }  
c = { 'uid': 106,   
 'login': 'ash',  
 'name' : 'Ashish Gulati'  
 }

Dict access**¶**

In [ ]:

u = c[’uid’] # Get an element  
c[’shell’] = "/bin/sh" # Set an element  
if c.has\_key("directory"): # Check for presence of an member  
 d = c[’directory’]  
else:  
 d = None  
  
d = c.get("directory",None) # Same thing, more compact

while loop**¶**

In [ ]:

a=1; b=10  
while a < b:  
 # Do something  
 a = a + 1

for loop**¶**

In [ ]:

for i in [3, 4, 10, 25]:  
 print i  
  
# Print characters one at a time  
for c in "Hello World":  
 print c  
  
# Loop over a range of numbers  
for i in range(0,100):  
 print i

No for (i=0; i<10; i++) loop

Functions**¶**

In [ ]:

# Return the remainder of a/b  
def remainder(a,b):  
 q = a/b  
 r = a - q\*b  
 return r  
  
# Now use it  
a = remainder(42,5) # a = 2

Returning multiple values**¶**

In [ ]:

def divide(a,b):  
 q = a/b  
 r = a - q\*b  
 return q,r  
  
x,y = divide(42,5) # x = 8, y = 2

Classes**¶**

In [ ]:

class Account:  
 def \_\_init\_\_(self, initial):  
 self.balance = initial  
 def deposit(self, amt):  
 self.balance = self.balance + amt  
 def withdraw(self,amt):  
 self.balance = self.balance - amt  
 def getbalance(self):  
 return self.balance

Creating instances**¶**

In [ ]:

a = Account(1000.00)  
a.deposit(550.23)  
a.deposit(100)  
a.withdraw(50)  
print a.getbalance()

Exceptions**¶**

In [ ]:

try:  
 f = open("foo")  
except IOError:  
 print "Couldn’t open ’foo’. Sorry."

Raising exceptions**¶**

In [ ]:

def factorial(n):  
 if n < 0:  
 raise ValueError,"Expected non-negative number"  
 if (n <= 1):  
 return 1  
 else:  
 return n\*factorial(n-1)

In [ ]:

factorial(-1)

File I/O**¶**

In [ ]:

f = open("foo","w") # Open a file for writing  
g = open("bar","r") # Open a file for reading

In [ ]:

f.write("Hello World")  
data = g.read() # Read all data  
line = g.readline() # Read a single line  
lines = g.readlines() # Read data as a list of lines

Formatted output**¶**

In [ ]:

for i in range(0,10):  
 f.write("2 times %d = %d\n" % (i, 2\*i)) # C printf-like formatting

Modules**¶**

Large programs can be broken into modules

In [ ]:

# numbers.py  
def divide(a,b):  
 q = a/b  
 r = a - q\*b  
 return q,r  
  
def gcd(x,y): g= y  
 while x > 0:   
 g = x  
 x = y%x  
 y = g   
 return g

In [ ]:

import numbers  
x,y = numbers.divide(42,5)  
n = numbers.gcd(7291823, 5683)

* import statment creates a namespace and executes a file in it
* all names accessible via modulename.funcname
* could be *unwrapped* to current namespace using from .. import ..

In [ ]:

from numbers import gcd, divide  
x,y = divide(42,5)  
n = gcd(7291823, 5683)

Standard modules**¶**

* Python comes pre-packaged with a large number of installed modules (Batteries Included)
* Other modules can be downloaded/installed to be used by programs