Python Introduction

Python – www.python.org

Open Source programming language with large Standard Library

Designed by Guido van Rossum (formerly @ CWI.NL, now @ Google)

Imperative, object-oriented, some functional programming

Interpreted (but compilation to Python byte code is possible)

Multi-platform

Used for scripting, coordination, web programming, ...

Third-party extensions

Python Interpreter

Prompt: >>>

Experiment interactively with

integers, floats, strings, tuples, lists, dictionaries, statements

>>> print 355 / 113, 355 % 113 # quotient and remainder
3 16

print not explicitly needed:

>>> float(355) / 113 # almost pi
3.1415929203539825
Example Python Program

```python
# Pay an amount exactly, using euro coins
amount = input('Amount between 0 and 500 euro cent: ')
for coin in 200, 100, 50, 20, 10, 5, 2, 1 :
    count = 0
    while amount >= coin :
        count = count + 1
        amount = amount - coin
    if count > 0 :
        print count, 'x', coin
```

Language: Syntax

- **Clean syntax** (unfortunately, = and == as in C/C++/Java)
- **Tuple assignment**: a, b = b, a+b
- **Conditional expression**: elif B else F
- **Block structure** is expressed by indentation level.
  ```python
  if condition1 :
    suite1
  elif condition2 :
    suite2
  else :
    suite3
  ```

Example Python Function Definition (pay_greedy.py)

```python
eurocoins = (200, 100, 50, 20, 10, 5, 2, 1) # tuple
oldn1coins = [250, 100, 25, 10, 5, 1] # list

def pay_greedy(amount, coins = eurocoins) :
    """Pay amount exactly, using coins greedily.
    Pre: 0 <= amount
    coins is decreasing sequence, containing 1
    Ret: bag of coins whose total value == amount
    ""
    result = {} # empty dictionary
    for coin in coins :
        result [ coin ], amount = divmod ( amount, coin )
    return result
```

Names, Objects, Values, and Types

Every **name** is bound to (refers to) an **object**.

Every **object** has an **identity**, a **type**, and a **value** of that type.

Names are not (meta)typed (cf. Pascal: const, type, var, procedure).

```python
name = "Python" # binds name to a string object
def name ( x ) : # binds name to a function object
    return x
class name : # binds name to a class object
    pass
```

© 2007–2009, T. Verheooff @ T.U.E.N.L.
Example Python Function Calls (pay_greedy.py)

```
print pay_greedy ( 388 ) # uses default value for param coins
print pay_greedy ( 388, oldnlcoins )
bag = pay_greedy ( coins = oldnlcoins, amount = 388 )

for coin in bag :
    print bag [ coin ], 'x', coin

for coin, freq in bag.items() :
    print freq, 'x', coin

for coin, freq in sorted ( bag.items() ) :
    if freq > 0 :
        print '%2d x %3d' % ( freq, coin )
```

© 2007–2009, T. Verhoeff @ TUE.NL 9 Python Introduction

Local versus global

```
pi = 3.14 # globally defined name

def circle_area1 ( r ) :
    return pi * r * r # uses global pi

def circle_area2 ( r ) :
    pi = 3.1416 # this defines a local pi
    return pi * r * r # uses local pi

def set_pi ( x ) :
    global pi
    pi = x # this affects the global pi
```

© 2007–2009, T. Verhoeff @ TUE.NL 10 Python Introduction

Immutable versus mutable objects

Numbers, strings and tuples are immutable: object value is constant.

Lists and dictionaries are mutable: object value can change.

```
n = 10 # n is initialized to a number object
n = n + 1 # n is bound to new number object

s = [ 3, 1, 2 ] # s is initialized to a list object
s.append(0) # value of list object bound to name s is modified
print s

t = s # ALIASING; use list(s) or s[:] to make a copy
t.sort() # value of list object is modified again
print s # s also turns out to be sorted
```

© 2007–2009, T. Verhoeff @ TUE.NL 11 Python Introduction

A Function That Bites (pay_greedyX.py)

```
def pay_greedyX ( amount, coins ) :
    """
    Pay amount exactly, using coins greedily.
    Pre: 0 <= amount, exactly payable (weaker pre)
    Ret: bag of coins whose total value == amount
    ""
    coins.sort()
    coins.reverse()
    result = { } # empty dictionary
    for coin in coins :
        result [ coin ], amount = divmod ( amount, coin )
    assert amount == 0, 'cannot pay amount exactly'
    return result
```

© 2007–2009, T. Verhoeff @ TUE.NL 12 Python Introduction
A Function That Bites (2)

```python
>>> myamount = 300

>>> mycoins = [ 1, 5, 10, 100, 250 ]  # old Dutch coins w/o 25

>>> pay_greedyX ( myamount, mycoins )  # (greedy not minimal)
{1: 0, 250: 1, 100: 0, 10: 5, 5: 0}

>>> myamount  # not changed
300

>>> mycoins  # changed!!!
[250, 100, 10, 5, 1]
```

Function Parameters Not Type( Checke)d: Can Do It Yourself

```python
def split ( n, b ) :
    """Determine coefficient c and exponent e of highest power
    of b in n, and the remainder r
    pre: 0 < n, 2 <= b
    ret: ( c, e, r ) with n = c * b^e + r, 0 < c < b, 0 < r < b^e
    ""
    assert type ( n ) == type ( 0 ), "split: param n not an int"
    assert type ( b ) == type ( 0 ), "split: param b not an int"
    assert 0 < n, "split: param n out of range (must be > 0)"
    assert 2 <= b, "split: param b out of range (must be >= 2)"
```

Overview of Classes

N.B. ‘old style’ versus ‘new style’ classes

Definition of class object; its ‘static’ attributes

Instantiation of class object; instance attributes; self

Inheritance

Exceptions, try ... except ... finally, raise

Class Definition Example (student.py)

```python
from datetime import date

class Person :
    """A class to represent persons"
    pcount = 0  # counts the instances

    def __init__ ( self, name, birthdate ) :  # constructor
        self.name = name
        self.birthdate = birthdate
        Person.pcount += 1

    def age ( self ) :
        """Returns age as timedelta in days"
        return date.today() - self.birthdate
```


**Class Instantiation Example (student.py)**

```python
print Person.pcount

# construct an instance
p = Person ( 'Tom', date(1958, 10, 24) )
print p.name, p.birthdate, p.age()
print Person.pcount

# construct another instance
q = Person ( 'Tim', date(1959, 10, 24) )
print q.name, q.birthdate, q.age()
print Person.pcount
```

**Class Inheritance Example (student.py)**

```python
class Student ( Person ) :
    """A class to represent students""
    scount = 0  # count the instances
    def __init__ ( self, name, birthdate, idnumber ) :
        Person.__init__(self, name, birthdate)
        self.idnumber = idnumber
        Student.scount += 1
print Person.pcount, Student.pcount, Student.scount

s = Student ( 'Sam', date(1987, 9, 19), 124866 )
print s.name, s.birthdate, s.age(), s.idnumber
print Person.pcount, Student.scount
```

**Exception Handling (exception.py)**

```python
a = input ( 'Give me an a: ' )

try :
    x = 1.0 / a
    print x
except ZeroDivisionError :
    print 'Cannot handle this problem'
    finally :
        # clean up
        print 'Done'
```

**Python Standard Library**

**Built-in Functions**

- `re`: Regular expressions
- `math`, `random`
- `datetime`
- `doctest`: To build in tests via doc strings
- `unittest`: Unit testing framework (a.k.a. PyUnit)
- `logging`
- `graphics`: turtle,...
**Built-in Functions: range(...)**

(No `import` statement needed)

```python
>>> range(10)  # list from 0 (default) to 10 (excl.)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> range(1, 10)  # list from 1 to 10 (excl.)
[1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> range(1, 10, 2)  # list from 1 to 10 (excl.) step 2
[1, 3, 5, 7, 9]
```

**Importing other modules**

```python
import math

print(math.pi)

from math import pi

print(163 * pi)

from math import exp

print(exp(163 * pi))
```

**Functional Programming Features**

```python
seq = range ( 1, 20 )

even = lambda ( n ) : n % 2 == 0
    # alternative form of function definition

map ( even, seq )  # apply even to each element of seq
map ( pow, seq, seq )  # apply built-in pow to each pair

[ (n*n) % 8 for n in seq ]  # list comprehension: squares mod 8

from operator import add, mul
reduce ( mul, seq, 1 )  # calculate product of elements in seq
```

**Gripes**

- No formatted input (cf. read in Pascal, scanf in C)
- Tom's personal gripes
References

Python Documentation

- Tutorial
- Language Reference
- Library Reference