Python Introduction

http://www.win.tue.nl/~wstomv/edu/python/

Tom Verhoeff

Technische Universiteit Eindhoven
Faculteit Wiskunde en Informatica
Software Engineering & Technology

Opmerkingen aan T.Verhoeff@TUE.NL

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

Practical Python

Many systems have a Python installation 'out of the box', including

Programming Tool IDLE (Integrated Development Environment)

Many editors support Python: syntax highlighting, execution

Python 2.x versus Python 3.x

Used in our programming education support system peach^3

Industrial support: Google, NASA, . . .

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

```
1  # Pay an amount exactly, using euro coins
2
3  amount = input ( 'Amount between 0 and 500 euro cent: ' )
4
5  for coin in 200, 100, 50, 20, 10, 5, 2, 1 :
6      count = 0
7
8      while amount >= coin :
9          count = count + 1
10         amount = amount - coin
11
12     if count > 0 :
13         print count, 'x', coin
```


Language: Syntax

- Clean syntax (unfortunately, `=` and `==` as in C/C++/Java)
- Tuple assignment: `a, b = b, a+b`
- Conditional expression: `E if B else F`
- Block structure is expressed by indentation level.

```
if condition1 :
    suite1
elif condition2 :
    suite2
else :
    suite3
```


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).

```
name = "Python"  # binds name to a string object
name = 42  # binds name to an integer object

def name ( x ) :  # binds name to a function object
    return x

class name :  # binds name to a class object
    pass
```


Example Python Function Definition (`pay_greedy.py`)

```
1  eurocoins = ( 200, 100, 50, 20, 10, 5, 2, 1 )  # tuple
2  oldnlcoins = [ 250, 100, 25, 10, 5, 1 ]  # list
3
4  def pay_greedy ( amount, coins = eurocoins ) :
5      """ Pay amount exactly, using coins greedily.
6          Pre: 0 <= amount
7               coins is decreasing sequence, containing 1
8          Ret: bag of coins whose total value == amount
9      """
10     result = { }  # empty dictionary
11
12     for coin in coins :
13         result [ coin ], amount = divmod ( amount, coin )
14
15     return result
```

Example Python Function Calls (pay_greedy.py)

```
17 print pay_greedy ( 388 )  # uses default value for param coins
18
19 print pay_greedy ( 388, oldnlcoins )
20
21 bag = pay_greedy ( coins = oldnlcoins, amount = 388 )
22
23 for coin in bag :
24     print bag [ coin ], 'x', coin
25
26 for coin, freq in bag.items() :
27     print freq, 'x', coin
28
29 for coin, freq in sorted ( bag.items() ) :
30     if freq > 0 :
31         print "%2d x %3d" % ( freq, coin )
```


Local versus global

```
1 pi = 3.14  # globally defined name
2
3 def circle_area1 ( r ):
4     return pi * r * r  # uses global pi
5
6 def circle_area2 ( r ):
7     pi = 3.1416  # this defines a local pi
8     return pi * r * r  # uses local pi
9
10 def set_pi ( x ) :
11     global pi
12     pi = x  # this affects the global pi
```


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
```


A Function That Bites (pay_greedyX.py)

```
1 def pay_greedyX ( amount, coins ) :
2     """ Pay amount exactly, using coins greedily.
3         Pre: 0 <= amount, exactly payable  (weaker pre)
4         Ret: bag of coins whose total value == amount
5     """
6     coins.sort()
7     coins.reverse()
8     result = { }  # empty dictionary
9
10    for coin in coins :
11        result [ coin ], amount = divmod ( amount, coin )
12
13    assert amount == 0, 'cannot pay amount exactly'
14    return result
```

A Function That Bites (2)

```
>>> 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

```
1  def split ( n, b ) :
2    """Determine coefficient c and exponent e of highest power
3       of b in n, and the remainder r
4       pre: 0 < n, 2 <= b
5       ret: ( c, e, r ) with n = c * b^e + r, 0<c<b, 0<=r<b^e
6    """
7    assert type ( n ) == type ( 0 ), "split: param n not an int"
8    assert type ( b ) == type ( 0 ), "split: param b not an int"
9    assert 0 < n, "split: param n out of range (must be > 0)"
10   assert 2 <= b, "split: param b out of range (must be >= 2)"
11   ....
```


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`)

```
1  from datetime import date
2
3  class Person :
4      """A class to represent persons"""
5      pcount = 0  # counts the instances
6
7      def __init__ ( self, name, birthdate ) :  # constructor
8          self.name = name
9          self.birthdate = birthdate
10         Person.pcount += 1
11
12     def age ( self ) :
13         """Returns age as timedelta in days"""
14         return date.today() - self.birthdate
```

Class Instantiation Example (`student.py`)

```
16 print Person.pcount
17
18 # construct an instance
19 p = Person ( 'Tom', date(1958, 10, 24) )
20
21 print p.name, p.birthdate, p.age()
22 print Person.pcount
23
24
25 # construct another instance
26 q = Person ( 'Tim', date(1959, 10, 24) )
27
28 print q.name, q.birthdate, q.age()
29 print Person.pcount
```


Class Inheritance Example (`student.py`)

```
31 class Student ( Person ) :
32     """A class to represent students"""
33     scount = 0  # count the instances
34
35     def __init__ ( self, name, birthdate, idnumber ) :
36         Person.__init__(self, name, birthdate)
37         self.idnumber = idnumber
38         Student.scount += 1
39
40 print Person.pcount, Student.pcount, Student.scount
41
42 s = Student ( 'Sam', date(1987, 9, 19), 124866 )
43 print s.name, s.birthdate, s.age(), s.idnumber
44 print Person.pcount, Student.scount
```


Exception Handling (`exception.py`)

```
1 a = input ( 'Give me an a: ' )
2
3 try :
4     x = 1.0 / a
5     print x
6 except ZeroDivisionError :
7     print 'Attempt to divide by 0'
8 else :
9     print 'Cannot handle this problem'
10 finally :
11     # clean up
12     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)

```
>>> 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

```
1 import math
2 print math.pi
3
4 from math import pi
5 print 163 * pi
6
7 from math import *
8 print exp ( 163 * pi )
```


Functional Programming Features

```
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 F1 in IDLE

- Tutorial
- Language Reference
- Library Reference