

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`

Open Source Programming Language with big Standard Library

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

Imperative, object-oriented, some functional programming

Interpreted, multi-platform

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

Programming Tool IDLE

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

```
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

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

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

Names, Objects, and Types

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

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

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

```
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

```
1 print pay_greedy ( 388 )   # uses default value for param coins
2
3 print pay_greedy ( 388, oldnlcoins )
4
5 bag = pay_greedy ( coins = oldnlcoins, amount = 388 )
6
7 for coin in bag :
8     print bag [ coin ], 'x', coin
9
10 for coin, freq in bag.items() :
11     print freq, 'x', coin
12
13 for coin, freq in sorted ( bag.items() ) :
14     if freq > 0 :
15         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

```
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]
```

Overview of Classes

Definition of class object; its 'static' attributes

Instantiation of class object; instance attributes; `self`

Inheritance

Exceptions, **`try ... except ... finally, raise`**

Class Definition Example

```
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    # create/initialize instance attribute
9         self.birthdate = birthdate    # another instance attribute
10        Person.pcount += 1
11
12    def age ( self ) :    # query
13        """Returns age as timedelta in days"""
14        return date.today() - self.birthdate
```

Class Instantiation Example

```
1 print Person.pcount
2
3 # construct an instance
4 p = Person ( 'Tom', date(1958, 10, 24) )
5
6 print p.name, p.birthdate, p.age()
7 print Person.pcount
8
9 # construct another instance
10 q = Person ( 'Tim', date(1959, 10, 24) )
11
12 print q.name, q.birthdate, q.age()
13 print Person.pcount
```

Class Inheritance Example

```
1 class Student ( Person ) :
2     """A class to represent students"""
3     scount = 0   # count the instances
4
5     def __init__ ( self, name, birthdate, idnumber ) :
6         Person.__init__(self, name, birthdate)   # inherited
7         self.idnumber = idnumber
8         Student.scount += 1
9
10 print Person.pcount, Student.pcount, Student.scount
11
12 s = Student ( 'Sam', date(1987, 9, 19), 124866 )
13 print s.name, s.birthdate, s.age(), s.idnumber
14 print Person.pcount, Student.scount
```


Exception Handling

```
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 )
```

References

Python Documentation

F1 in IDLE

- Tutorial
- Language Reference
- Library Reference