

PYTHON

Gopas

What is PYTHON ?

- **Modern programming language**
- Author Guido van Rossum, university of Amsterdam
- Comes from **C**, **C++** and **Modula-3**
- Is platform **independent** (UNIX, Windows, MacOS X, OS/2, etc.)
- Is very **productive** -> fast application prototyping
- Well integrated with other languages C, C++ (**CPython**), .NET (**Iron Python**)
a JAVA (**Jython**) !!!
- Lots of available modules
 - Multi paradigm: simple procedural programming, object-orientation and functional programming.
- Easy to learn :-)

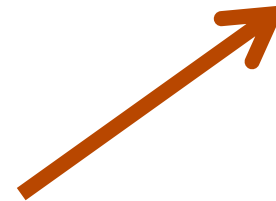
Structure of source code

```
int c;  
float a,b;  
a=2.584;  
c=1;  
while (c<100) {  
    ++c;  
    b=a*c;  
}
```

C



Pascal



```
var c: integer;  
var a,b: real;  
a:=2.584;  
c:=1;  
while (c<100) do begin  
    c:=c+1;  
    b:=a*c;  
end
```

PYTHON



```
a=2.584  
c=1  
while c<100:  
    c+=1  
    b=a*c
```

Python sources

- . Homepage <http://www.python.org>
- . Official documentation <http://docs.python.org>
- . **Two branches** Python, version **2** (ended) and **3**.

Python books



Mark Lutz:
Programming Python
(standard textbook)



Alex Martelli:
Python in a Nutshell
(good reference)



Alex Martelli:
Python Cookbook

Python

Python shell IDLE

Python REPL – Read Evaluate Print Loop

**Python IDE pyDev for Eclipse
PyCharm
Visual Code Studio
Komodo**

Comments

My first program

a=2.584

a is variable

c=1

while c<100:

c+=1

b=a*c

Data Types – Built-In Types

- integer – `a=10`, `b=0xa`, `c=012`
- long integer – `a=88888L`, `b=-777777L`
- float – `a=1.0`, `b=-1.5e3`
- complex number – `a=1+2j`, `b=1.5-2.58j`

(modules *decimal* and *fractions*)

- boolean – `True`, `False`
- string – `a="Hi"`, `b='Bye'`, `c=""Hello""`
- tuple – `a=("one","two",3,4)`
- list – `a=["one","two",3,4]`
- set – `a={"one","two",3,4}`
- dictionary – `a={"one":1, "two":2}`

Mutable and immutable data types

Immutable types : integer, long integer, float, complex number, string, tuple.

Mutable types : list, set, dictionary.

Operations on Numbers

+	...	add
-	...	minus
*	...	multiplication
**	...	exponent
/	...	division
%	...	modulo, operator for string format

Python accepts shortcuts: **+=**, **-=**, ***=**, ...

Comparison

== ... value equality

!= ... unequal

< ... less then

> ... greater then

<= ... less then equal

>= ... greater then equal

in ... in sequence (***not in***)

is ... object equality (***is not***)

Boolean operations

and ... Logical and

or ... Logical or

not ... Negotiation

Binary operators

& ... AND

| ... OR

^ ... XOR

~ ... NOT

Numbers

1, -10, 458

integers

0xff12, -0x14

integers (hexadecimal)

0b1100

integers (binary)

0o457

integer (octal)

1.0, -5.5e3

floating-point

9999999L

long integers (2.x)

1j, 5+4j, 2+5.4j

complex

Math functions *abs*, *min*, *max*, *round*, . . .

More of them are in module *math*.

Math functions for complex numbers in module *cmath*.

Strings

`r = 'String with new line at the end\n'`

`r = " String with new line at the end \n"`

`r = """Three single quotes"""`

`r = """"Three double-quotes""""`

`r = "raw string no escape treated\n"`

`r = u"unicode string\n"`

Work with strings through tuples. Basic functions *chr*, *ord*, *len*.
More in module *string*

Strings – escape sequences

\n ... New line

\t ... Tab

\\ ... Backslash

\' ... Single quote

\\" ... Double quote

\nnn ... Octal ASCII char

\xnn ... Hex ASCII char

Strings – UNICODE

`r = u“String UNICODE. We can use escape sequences.”`

`unicode ('ščšč','iso8859-2')`

`unicode ('ščšč','iso8859-1')`

`unicode ('ščšč','ascii')`

`'ČŠČČŘ'.decode('iso8859-2')` ... method ***decode*** makes same as function ***unicode***

`'příšera'.encode('utf-8')`

Strings – conversion

str (object) - string

int (string) – number of type *integer*

long (string) – number of type *long integer (2.x)*

float (string) – number of type *float*

complex (string) – number of type *complex*

Strings – format

formatString % object

formatString % (object1, object2, ...)

formatString % dictionary

a=10

print("Variable has value %d" % a)

String format operators :

s ... String presentation of object

d ... integer

f ... float

"Value is: {0}".format(123)

Strings – more operations

Methods of object *string* :

find (s,substring)

join (seznam)

lower(s)

upper(s)

replace(s,substring,replace)

split(s,separator)

strip(s)

lstrip(s)

rstrip(s)

.....

Lists (arrays)

x=[] ... Constructs empty list

x=[1,2+3j, "next",4] ... Creates and fills list

[1,2] + ["three", "four"] ... **[1,2, "three", "four"]**

2 * [1,2] ... **[1,2,1,2]**

list ('gopas') ... **['g','o','p','a','s']** ... List from sequence

list ([1,85,96]) ... **[1,85,96]** ... List from sequence

len (['g','o','p','a','s']) = 5 ... Number of items

List – access to items, slicing ...

x=[1 , 2 , 3 , 4 , 5 , 6]

x[2] ... 3

x[-3] ... 4

x[1:4] ... [2,3,4]

x[:-2] ... [1,2,3,4]

x[3:] ... [4,5,6]

x[:] ... [1,2,3,4,5,6] (*copy*)

Lists - modifications

x=[1,2,3,4,5,6]

y=["two","three","four"]

x[1]="two" ... [1,"two",3,4,5,6]

x[1] = y ... [1, ["two", "three", "four"], 3, 4, 5, 6]

x[1:3] = y ... [1,"two","three","four",4,5,6]

x.append(7) ... [1,2,3,4,5,6,7]

x.insert(0,"zero") ... ["zero",1,2,3,4,5,6]

x.remove(5) ... [1,2,3,4,6]

del x[1] ... [1,3,4,5,6]

del x[2:4] ... [1,2,5,6]

Lists – more operations

x=[2,4,1,5,6,3,3]

x.sort() ... [1,2,3,3,4,5,6]

3 in x ... True

2 not in x ... False

min(x) ... 1

max(x) ... 6

x.index(1) ... 2 (index of item in list)

x.count(3) ... 2 (number of item in list)

Tuples

x=() ... Empty tuple

x=(1,) ... One item tuple

x=(1,2+3j,"next",4) ... Creates tuple

(1,2) + ("three","four") ... (1,2,"three","four")

2 * (1,2) ... (1,2,1,2)

tuple ('gopas') ... ('g','o','p','a','s') tuple from sequence

tuple ([1,85,96]) ... (1,85,96) ... Tuple from list

len (('g','o','p','a','s')) ... 5 ... Number of items

Tuples - access

x=(1,2,3,4,5,6)

x[2] ... 3

x[-3] ... 4

x[1:4] ... (2,3,4)

x[: -2] ... (1,2,3,4)

x[3:] ... (4,5,6)

Sets

x=set() ... Empty set

x={1,2,3} ... Creates and fills set

len(x) ... Number of items in set

x.add(4) ... Adds next item to set

x.remove(4) ... Removes item from set (if not present, creates exception)

x.discard(4) ... Removes item from set

x.clear() ... Clears set

x.copy() ... *shallow copy*

Dictionary

Associative array, hash array

x={} ... Creates empty dictionary

x={"cerveny":"red","zeleny":"green"} ... Creates dictionary

x["bily"]="white" ... Adds key to dictionary

len(x) ... 3 ... Number of keys

x.keys() ... ["cerveny","zeleny","bily"] ... List of keys

x.values() ... ["red", "white", "green"] ... List of values

x.items() ... [('cerveny', 'red'), ('bily', 'white'), ('zeleny', 'green')]

x.clear() ... Clears dictionary

x.copy() ... (*shallow copy*)

Reference and copy I.

```
a=[[0,1],2,3,4]
```

```
b=a
```

```
b[0][1]="one"
```

```
print (a,b)
```

Reference and copy II.

```
a=[[0,1],2,3,4]
```

Shallow copy (container objects, creates references for source objects) :

```
b=a[:]
b.append(5)
print a,b
b[0][1]= "one"
print (a,b)
```

Deep copy (creates object and recursively copies and creates new objects) :

```
import copy
b=copy.deepcopy(a)
b.append(5)
print a,b
b[0][1]= "one"
print (a,b)
```

Conditionals: if-else Statements

if (*condition*):

block of statements 1

else:

block of statements 2

More if-elif-else

```
if (condition1):  
    block of statements 1  
elif (condition2):  
    block of statements 2  
elif (condition3):  
    block of statements 3  
.....  
else:  
    block of statements
```

While loop

```
while (condition):  
    block of statements 1  
else:  
    block of statements 2
```


For loop

for *variable in sequence*:
 block of statements 1
else:
 block of statements 2

Commands : range, break a continue

List comprehension

[<expression> for <variable> in <collection> if <condition>]

Can be written as :

L = []

for variable in collection:

if condition:

L.append(expression)

Example :

[n for n in range(10) if n%2==0]

Functions and procedures

```
def name (parameter1, parameter2, ...):  
    """Documentation string"""  
    body  
    return value
```

Functions and procedures

```
def greeting ( ):  
    """Prints greeting"""  
    print ("Hello")
```

```
greeting ( )
```

```
def add (a,b):  
    """Sum of two numbers"""  
    c=a+b  
    return c
```

```
x=add (3,2)  
print (x)
```

Lambda Expressions

```
toUpper = lambda r : r.upper()
```

```
print (toUpper('hello, how are you ?'))
```

Functions and procedures – default parameters

```
def exp (z,e=2):  
    """Counts exponent"""  
    x=z  
    while e>1:  
        x=x*z  
        e=e-1  
    return x  
  
print (exp(2,3))  
print (exp(2))
```

Functions and procedures – named parameter passing

```
def exp (z,e=2):  
    """Counts exponent"""  
    x=z  
    while e>1:  
        x=x*z  
        e=e-1  
    return x  
  
print exp (2,3)  
print exp (e=3,z=2)
```

Functions and procedures – variable list of parameters

```
def maximum (*cisla):  
    """Counts max"""  
    m=cisla[0]  
    for n in cisla[1:]:  
        if n>m:  
            m=n  
    return m  
  
print (maximum(1,5,4,2))  
print (maximum(1,5,4,2,23,23))
```


Functions and procedures – variable list of parameters

```
def example (**params):  
    print (params)
```

```
example (a=1,b=2,c=3)
```

Functions and procedures – mutable objects as arguments

```
def example (n, list1, list2):  
    """“Mutable objects”"""  
    list1.append('Black')  
    list2=[3,2,1]  
    n+=1
```

```
a=1  
b=['Joe']  
c=[1,2,3]  
example(a,b,c)  
print(a,b,c)
```

Functions and procedures – namespaces, scope of vars

```
def example ():  
    """Namespace"""  
    a=1  
    b=2
```

```
a=10  
b=20  
example()  
print(a,b)
```

Functions and procedures – namespaces, scope of vars

```
def example ():  
    """Local/global variables"""  
    print(a)  
    b=2
```

```
a=10  
b=20  
example()  
print(a,b)
```

Global vars are automatically read-only!!!

Functions and procedures – sequences

```
def add (a,b):  
    return (a+b)
```

```
first=[1,2,3]  
second=[4,5,6]
```

```
print (map(add,first,second))
```

map: return a list containing the result of some function applied to each object in a collection

Functions and procedures – work with sequences

```
def even (a):  
    return not (a%2)  
  
numbers=range(20)  
  
print (filter(even,numbers))
```

filter: retain only elements for which the function returns True.

Modules I.

modul.py



```
"""Test module"""  
a=10  
def sum (x,y):  
    """Sum of numbers"""  
    s=x+y  
    return s
```

```
import modul
```

```
print (modul.a)  
result=modul.sum(2,3)  
print (result)
```

Modules I.

modul.py



```
"""Test module"""  
a=10  
def sum (x,y):  
    """Sum of numbers"""  
    s=x+y  
    return s
```

```
import modul as m
```

```
print (m.a)  
result=m.sum(2,3)  
print (result)
```


Modules II.

modul.py



```
"""Test module"""  
a=10  
def sum (x,y):  
    """Sum of numbers"""  
    s=x+y  
    return s
```

```
from modul import *
```

```
print (a)  
result=sum (2,3)  
print (result)
```

Modules III. (protected names)

modul.py

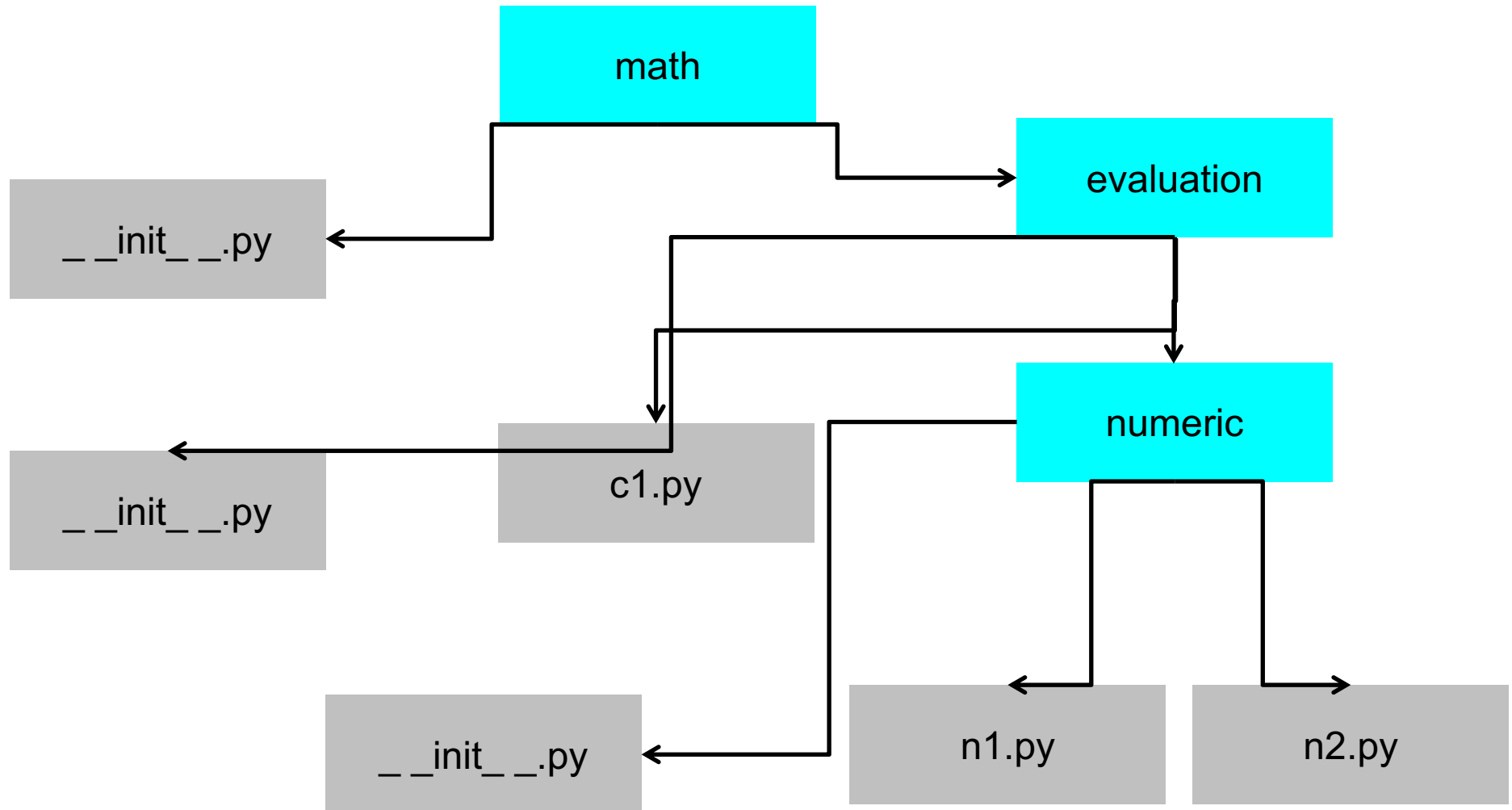


```
"""Test module"""  
a=10  
_b=20  
def sum (x,y):  
    """Sum of numbers"""  
    s=x+y  
    return s
```

```
from modul import *
```

```
print (a)  
print (_b)  
result=sum (2,3)  
print (result)
```

Packages



Errors and exceptions I.

1) We ignore errors

2) Watch results of all I/O operations
(just like in Pascal, C, ...)

3) Exceptions handling
(like in JAVA, PYTHON, RUBY, ...)

Errors and exceptions II.

function getFromServer

try *execute following code*

 openNetworkConnection....

 sendHTTPrequest....

 closeNetworkConnection....

except *if any error occurred, trap exception*

Errors and exceptions III.

try:

print (1/0)

except ZeroDivisionError:

print ("Division by zero !!!")

Errors and exceptions IV.

try:

```
file = open("data.txt","r")
```

except IOError as ex:

```
if (ex.errno==2): # file not exists
```

```
    print ("File doesn't exists")
```

else:

```
    print ("File exists: %d : %s" % (ex.errno, ex.strerror))
```

Errors and exceptions V.

```
def division (a,b):  
    if b==0:  
        raise ZeroDivisionError, " Division by zero"  
    return 0  
v=a/b  
return v  
  
try:  
    v=division(4,0)  
    print (v)  
except ZeroDivisionError,text:  
    print (text)
```


Errors and exceptions VI.

```
try:  
    i=10/0  
except:  
    print('Caught exception')  
else:  
    print('No exception')  
finally:  
    print('Finishing')
```

Errors and exceptions VI.

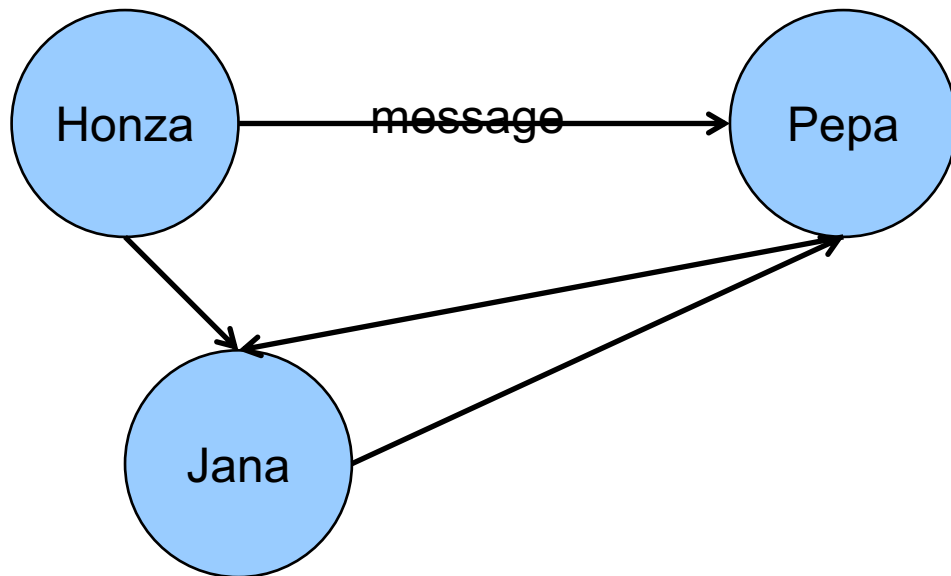
Exception

+-- SystemExit	+-- TypeError
+-- StopIteration	+-- AssertionError
+-- StandardError	+-- LookupError
+-- KeyboardInterrupt	+-- IndexError
+-- ImportError	+-- KeyError
+-- EnvironmentError	+-- ArithmeticError
+-- IOError	+-- OverflowError
+-- OSError	+-- ZeroDivisionError
+-- WindowsError	+-- FloatingPointError
+-- EOFError	+-- ValueError
+-- RuntimeError	+-- UnicodeError
+-- NotImplementedError	+-- UnicodeEncodeError
+-- NameError	+-- UnicodeDecodeError
+-- UnboundLocalError	+-- UnicodeTranslateError
+-- AttributeError	+-- ReferenceError
+-- SyntaxError	+-- SystemError
+-- IndentationError	+-- MemoryError
+-- TabError	+---Warning
	+-- UserWarning
	+-- DeprecationWarning
	+-- PendingDeprecationWarning
	+-- SyntaxWarning
	+-- RuntimeWarning
	+-- FutureWarning

Object oriented programming (OOP)

Goal of OOP is make programming closer to real world problem.

OO program is network of connected and communicating objects.



Object oriented programming (OOP)

Classes:

User defined types of objects (including their methods, attributes, relations to other objects).

Can be instantiated into an object / is a 'blueprint' that describes how to build an object.

Python does not enforce OOP (unlike Java), but we need to understand at least what is going on.

Class definitions contain methods (which are functions defined in the class' scope), class attributes, and a docstring.

Object oriented programming (OOP)

Basic terms:

- . Class**
- . Instance**
- . Instance variable, method**
- . Atribut**
- . Inheritance**
- . Encapsulation**
- . Polymorphism**

Object oriented programming (OOP)

```
class MyClassName:  
    "Doc string"  
    body
```

```
instance=MyClassName()
```

Object oriented programming (OOP)

```
class Person:  
    pass
```

```
pepa=Person()  
pepa.name="Josef"  
pepa.prijmeni="Novak"
```

```
lojza=Person()  
lojza.name="Alois"  
lojza.prijmeni="Novy"
```

Object oriented programming (OOP)

(instance variables, methods)

```
class Person:
    def printall(self):
        print ("Name : %s, age : %d" % (self.name,self.age))

pepa=Person()
pepa.name="Josef"
pepa.age=20
pepa.printall()
```


Object oriented programming (OOP)

(special/magic methods)

Python has rich set of special methods which implement several operations or syntax constructs:

- . creation/destroy of objects**
- . arithmetic operations**
- . logic operations (comparisions)**
- . work with sequences**
- . work with attributes**
-**

Object oriented programming (OOP)

(magic methods)

```
class Person:
    def __init__(self,name='',age=0):
        self.name=name
        self.age=age
    def printall(self):
        print ("Name : %s, age : %d" % (self.name,self.age))

pepa=Person("Josef",20)
pepa.printall()
```

Object oriented programming (OOP)

(magic methods)

```
class Person:
    def __init__(self,name='',age=0):
        self.name=name
        self.age=age
    def __str__(self):
        return(self.name)
    def printall(self):
        print ("Name : %s, age : %d" % (self.name,self.age))

pepa=Person("Josef",20)
pepa.printall()
print(str(pepa))
```

Object oriented programming (OOP)

(magic methods)

```
class Person:  
    def __init__(self,name,age):  
        self.name=name  
        self.age=age  
    def __str__(self):  
        return(self.name)  
    def __gt__(self,other):  
        if (self.age>other.age):  
            return True  
        return False  
    def printall(self):  
        print ("Name : %s, age : %d" % (self.name,self.age))
```

```
pepa=Person("Josef",20)  
lojza=Person("Alois",19)  
print(pepa>lojza)
```

Object oriented programming (OOP)

(magic methods)

```
class Person:
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def __str__(self):
        return(self.name)
    def __gt__(self,other):
        if (self.age>other.age):
            return True
        return False
    def __add__(self,other):
        return self.age+other.age
    def printall(self):
        print ("Name : %s, age : %d" % (self.name,self.age))
```

```
pepa=Person("Josef",20)
lojza=Person("Alois",19)
print(pepa+lojza)
```

Object oriented programming (OOP)

(class variables)

```
class Person:
```

```
    Person_id=1
```

```
    def __init__(self,name,age):
```

```
        self.name=name
```

```
        self.age=age
```

```
        self.cid=Person.Person_id
```

```
        Person.Person_id+=1
```



```
self.__class__.Person_id
```

```
    def printall(self):
```

```
        print ("Name : %s, age : %d, id : %d" % (self.name,self.age,self.cid))
```

```
pepa=Person("Josef",20)
```

```
lojza=Person("Alois",19)
```

```
pepa.printall()
```

```
lojza.printall()
```

Object oriented programming (OOP)

(class methods)

```
class Person:
```

```
    Person_id=1
```

```
    def __init__(self,name,age):
```

```
        self.name=name
```

```
        self.age=age
```

```
        self.cid=Person.Person_id
```

```
        Person.Person_id+=1
```

```
    def resetPerson(cls):
```

```
        cls.Person_id=1
```

```
    resetPerson=classmethod(resetPerson)
```

```
    def printall(self):
```

```
        print ("Name : %s, age : %d, id : %d" % (self.name,self.age,self.cid))
```

```
pepa=Person("Josef",20)
```

```
pepa.resetPerson()
```

```
lojza=Person("Alois",19)
```

```
pepa.printall()
```

```
lojza.printall()
```

Object oriented programming (OOP)

(class methods)

```
class Person:
    Person_id=1

    def __init__(self,name,age):
        self.name=name
        self.age=age
        self.cid=Person.Person_id
        Person.Person_id+=1

    @classmethod
    def resetPerson(cls):
        cls.Person_id=1

    def printall(self):
        print ("Name : %s, age : %d, id : %d" % (self.name,self.age,self.cid))

pepa=Person("Josef",20)
pepa.resetPerson()
lojza=Person("Alois",19)

pepa.printall()
lojza.printall()
```



Decorator

Object oriented programming (OOP)

(inheritance)

```
class Person:
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def printall(self):
        print ("Name : %s, age : %d" % (self.name,self.age))
```

```
class Student(Person):
    def __init__(self,name,age,school):
        Person.__init__(self,name,age)
        self.school=school
    def printall(self):
        Person.printall(self)
        print("School : %s" % self.school)
```

```
pepa=Student("Josef",20,"basic")
pepa.printall()
```

Object oriented programming (OOP)

(private names)

```
class trida:
    def __init__(self):
        self.x=1
        self.__y=2
    def printall(self):
        print self.x
        print self.__y
```

```
t=trida()
t.printall()
print t.x
print t.__y
```

I/O operations I.

`myfile=open ("/tmp/myfile.txt","r")` ... Open file for read
`myfile.close()` ... Close file

File open modes:

r ... read

w ... write

a ... append

I/O operations II.

Read from file

```
myfile=open ("/etc/passwd","r")
```

```
count=0
```

```
while myfile.readline() != "":
```

```
    count+=1
```

```
myfile.close()
```

```
print "In system are %d users" % count
```

```
import string
```

```
myfile=open ("/etc/passwd","r")
```

```
rows=myfile.readlines()
```

```
for a in rows:
```

```
    print string.rstrip(a)
```

```
myfile.close()
```

I/O operations III.

Write to file

```
myfile=open ("/tmp/myfile.txt","w")  
myfile.write("First row\n")  
myfile.write("Second row\n")  
myfile.close()
```

```
myfile1=open ("/etc/passwd","r")  
rows=myfile1.readlines()  
myfile1.close()  
myfile2=open ("/tmp/passwd.bak","w")  
myfile2.writelines(rows)  
myfile2.close()
```

I/O operations IV.

operating system

```
import os
print os.name ..... nt, posix, mac
print os.getcwd() ..... /home/pepa
print os.listdir('/tmp') ..... List of files from directory
os.chdir('/tmp') ..... Directory change

print os.path.join('home','honza') ..... home/honza

print os.path.exists('/tmp') ..... Finds if file exists

print os.path.isfile('/etc/passwd') ..... Finds if myfile regular file
print os.path.isdir('/etc/passwd') ..... Finds if myfile directory
```

User input

```
a = input("Input number a : ")  
b = input("Input number b : ")  
print "Sum %d + %d = %d" % (a,b,a+b)
```

Method: *raw_input*

Stdin, stdout, stderr

In module *sys* exists three file objects :

sys.stdin ... standard input

sys.stdout ... standard output

sys.stderr ... standard error output

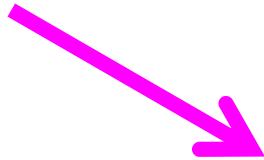
Stdin implements methods ***readline***, ***readlines*** a ***xreadlines***

Outputs implement ***write*** a ***writelines***

Persistance of objects

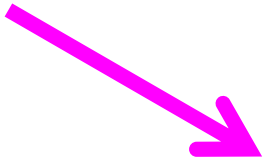
(with files)

Write to file



```
import pickle
a="This is my text"
b=[1,2,4,5,6]
myfile=open("/tmp/stav","w")
pickle.dump(a,myfile)
pickle.dump(b,myfile)
myfile.close()
```

Read from file



```
import pickle
myfile=open("/tmp/stav","r")
a=pickle.load(myfile)
b=pickle.load(myfile)
myfile.close()
print a
print b
```

Modul shelve

```
import shelve
adresar=shelve.open("/tmp/adresy")
adresar["policie"]=["Statni policie","158"]
adresar["hasici"]=["Hasicky sbor","150"]
adresar.close()
```

```
import shelve
adresar=shelve.open("/tmp/adresy")
print adresar["policie"]
print adresar["hasici"]
adresar.close()
```

Scripts I.

```
#!/usr/bin/python
```

```
def main():  
    print "This is our script !!!"
```

```
if __name__ == '__main__':  
    main()
```

Scripts II.

Arguments from command line

```
#!/usr/bin/python
```

```
import sys
```

```
def main():
```

```
    print sys.argv
```

```
main()
```

Scripts III.

modul getopt

```
#!/usr/bin/python
```

```
import sys,getopt
```

```
def main():
```

```
    (options,agruments)=getopt.getopt(sys.argv[1:], "a:b:c")
```

```
    print (options)
```

```
    print (agruments)
```

```
main()
```

```
./pokus.py -a1 -b 2 -c arg1 arg2
```

Network communication

Python is supported with number of modules for socket oriented communications and communication protocol implementations:

- . socket**
- . http**lib
- . ftp**lib
- . url**lib
- . smtp**lib
- . nntp**lib
- . pop**lib
- . imap**lib
-**

GUI

1) dialog, cdialog, xdialog

2) wxPython

3) Tkinter

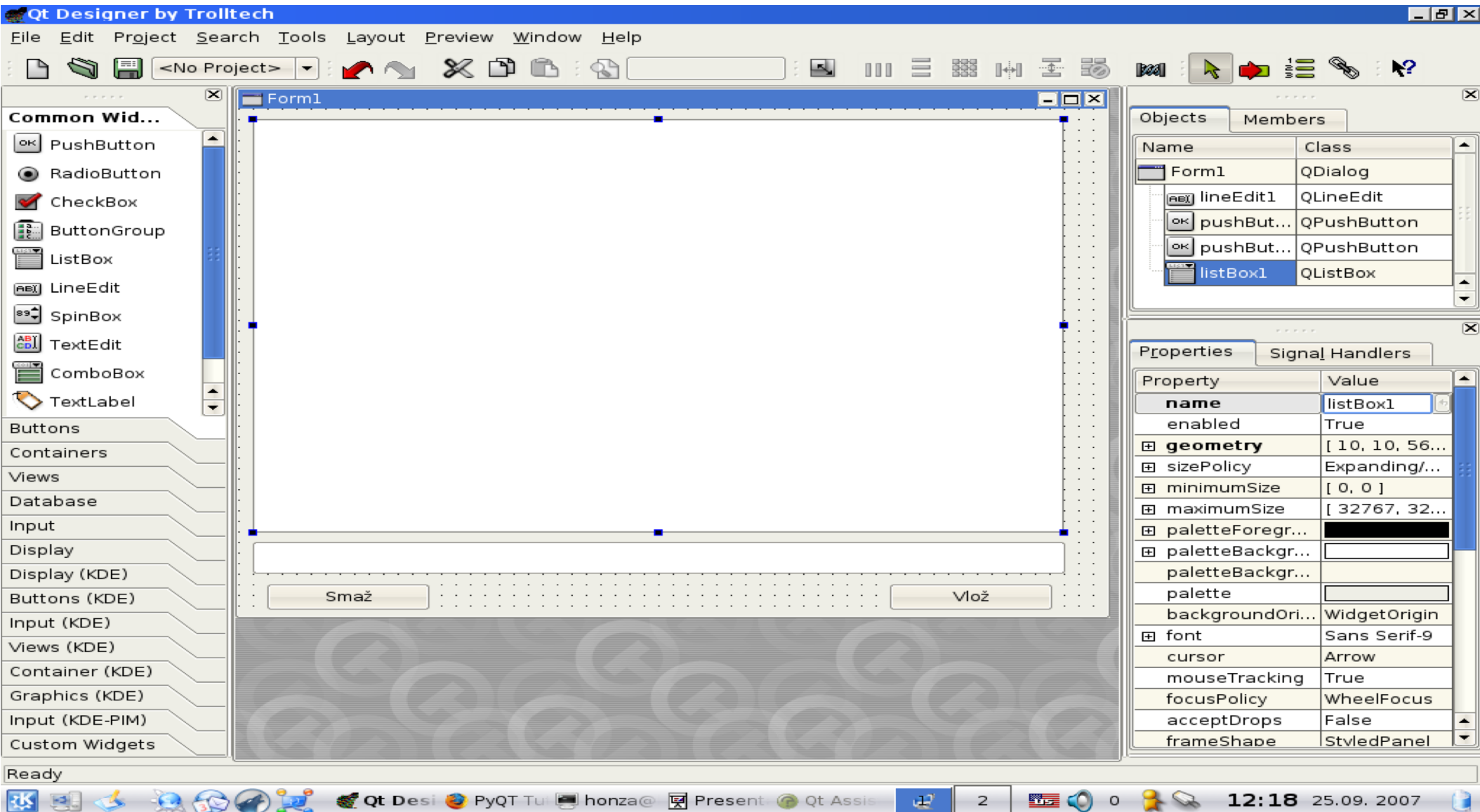
4) PyGTK

5) PyQT !!!!!

6) PyGame

GUI

Qt and PyQt – Qt Designer



Work with databases

Python defines Python Database API Specification v2.0

Relational databases are the most widely used type of database, storing information as tables containing a number of rows.

Example SQLite

Work with databases

```
import sqlite3
```

```
conn=sqlite3.connect("phones.sqlite")
```

```
cursor=conn.cursor()
```

```
cursor.execute("select * from phones")
```

```
for record in cursor.fetchall():
```

```
    print("Name : %s, phone number : %s" %(record[0],record[1]))
```

```
conn.close()
```

Work with databases

```
import sqlite3
```

```
conn=sqlite3.connect("phones.sqlite")
```

```
cursor1=conn.cursor()
```

```
cursor2=conn.cursor()
```

```
cursor1.execute("insert into phones values ('Police','158')")
```

```
conn.commit()
```

```
cursor2.execute("select * from phones")
```

```
for record in cursor2.fetchall():
```

```
    print("Name : %s, cislo : %s" %(record[0],record[1]))
```

```
conn.close()
```

PyPI

- The Python Package Index is a repository of software for the Python programming language.
- There are currently **48184** packages here
- Management tool *pip*.