Python programmation

Jarmo Hietala: 2020-07-22

Capitulo 1

Linguage Python

Python es un linguage de programmation facile a apprender, moderne et effective. Le linguage es orientate a objectos e ha syntaxe de alte nivello. Il ha typage dynamic e character interpretate.

Le interprete interactive

Nos initia le interprete de Python per le commando

```
python
```

Al comencio le interprete scribe a nos su version de programma e attende nostre instructiones.

```
# python
Python 3.8.3 (default, May 29 2020, 00:00:00)
[GCC 10.1.1 20200507 (Red Hat 10.1.1-1)] on linux
>
```

Operatores arithmetic

Nos trova que le interprete de Python functiona ben como calculator. Operatores arithmetic basic es `+`, `-`, `*` et `/`. Le operatores `*` et `/` ha plus alte

prioritate que le operatores `+` et `-`, ma nos pote scriber expressiones in parentheses `(…)` pro cambiar le ordine de calculation. Le operator a elevar in potentia es `**`.

```
> 2 + 2
4
> 50 - 5 * 6
20
> 3 * (5 - 2)
9
> 2 ** 8
256
```

Le division (`/`) sempre resulta in numero decimal de classe `float`. Le operatores `//` et `%` da nos le quotiente e le residuo de division de numeros integre. Le numeros integre ha le typo `int`, in altere parolas, illes es de classe `int`.

```
> 8 / 5
1.6
> 11 // 4
2
> 11 % 4
3
```

Valores veritate

Le valores veritate es `False` e `True`. Operationes boolean basic es `and`, `or`, e `not` (tabella 1).

Tabella 1. Operationes boolean.

not	
False	True
True	False

and	False	True
False	False	False
True	False	True

or	False	True
False	False	True
True	True	True

Per exemplo,

```
> not False
```

```
True
> False and True
False
> False or True
True
```

Operatores a comparar es `<`, `>`, `==`, `<=`, `>=`, e `!=` (tabella 2).

Tabella 2. Operatores a comparar.

Operator	Signification
`<`	minor que
`>`	major que
`==`	equal a
`<=`	minor que o equal a
`>=`	major que o equal a
`!=`	inequal a

Per exemplo,

```
> 3 < 4 < 5                    #          __
True                           #      ___( o)>
> 10 + 10 != 20                #      \ <_. )         >(.)__
False                          #       `---'           (___/
```

Sequentias de characteres

Sequentias de characteres pote esser representate inter virgulettas singule (`'…'`) o inter virgulettas duple (`"…"`). Le character a escappar es barra oblique inverse (\).

```
> 'Python' == "Python"
True
> 'Monty\'s' == "Monty's"
True
```

Le operatores `+` et `*` adjunge e repete sequentias de characteres.

```
> 'i' + 3 * 'nte'
'intentente'
```

Quando inter sequentias de characteres existe solmente spatio blanc, le interprete adjunge le sequentias.

```
> "inter"   "lingua"
'interlingua'
```

Il es possibile a trenchar un sequentia de characteres per su indices. Le prime character ha le indice zero. Le indice negative calcula de fin.

```
> t = 'Python'
> t [2:6], t [4:], t [-3:-1], t [-2:], t [-1]
('thon', 'on', 'ho', 'on', 'n')
```

Listas

Un *lista* es un collection de valores scribite inter parentheses quadrate `[…]`. Nos defini un lista de nomines de numeros cardinal:

```
numeros = [ 'zero', 'un', 'duo', 'tres', 'quatro', 'cinque',
 'sex', 'septe', 'octo', 'nove', 'dece' ]
```

Nos trova le *elementos* per le indices. Indices de un lista comencia de zero. Le `[]` es un lista vacue, e `[1]`, per exemplo, es un lista con un elemento.

```
> numeros [0]
'zero'
> numeros [5]
'cinque'
```

Nos anque pote trenchar le lista per le indices.

```
> numeros [3:7]
['tres', 'quatro', 'cinque', 'sex']
```

Dictionarios

Un dictionario consiste de pares de forma *clave*:*valor* scribite inter parentheses crispe `{…}`.

```
nums = {
  'un': 1, 'duo': 2,   'tres': 3, 'quatro': 4, 'cinque': 5,
  'sex':6, 'septe': 7, 'octo': 8, 'nove': 9,   'dece': 10 }
```

Nos trova le valor per su clave.

```
> nums ["cinque"]
5
```

Tuplas

Un *n-tupla* es un sequentia de n valores. In Python nos scribe tuplas in parentheses `(…)`, per exemplo, `t = (1,2)`, o nos pote omitter le parentheses: `t = 1,2`. Le `()` es un 0-tupla, e `(2,)` es un 1-tupla.

Quando le formas es favorabile, il es possibile a dispacchettar le elementos de un tupla, per exemplo `(a,b) = t`.

```
> p = 1,2
> a,b = p
> a
1
> b
2
```

Variabiles e definitiones de functiones

Nos dice que le *sequentia Fibonacci* es le numeros 0 et 1 e sempre le summa de duo ultime numeros.

Nos pote definir un function `fibonacci (n)`, que calcula le `n` prime numeros de un sequentia Fibonacci:

```
def fibonacci (n):
  result = []
  a,b = 0,1
  while (len (result) < n):
    result.append (a)
    a,b = b, a+b
```

```
  return result
```

Nos voca le function per valor `n = 10`. Le resultato es un lista de 10 elementos:

```
> fibonacci (10)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Hic le predefinite function `len (xs)` retorna le longor de lista (o sequentia) `xs`. Le listas ha un *methodo* `append (a)`, que adjunge le elemento `a` a un lista.

```
> len ('interlingua')
11
> xs = []
> xs.append (4)
> xs
[4]
```

Programma graphic con modulo `tkinter`

Nos nunc es preste a scriber nostre prime programma graphic. Pro isto nos usa le modulo venerabile `tkinter`. Nos salva le texto in file `hello.py`.

```
from tkinter import *

root = Tk ()
w = Label (root, text="Salute Mundo!")
w.pack ()
root.mainloop ()
```

Nos lancea le programma per commando `python hello.py`. In figura 1 nos vide qual es su aspecto.

Figura 1. Nostre prime programma graphic.

Importar modulos

In Python *modulos* son collectiones de *nomines*, id es, functiones e structuras de datos. Quando nos vole usar iste nomines, nos debe *importar* le modulo correspondente.

Il existe differente manieras a importar un modulo. Si le nomine de un modulo es `modulo` e illo ha functiones `f1`, `f2`, ..., nos pote importar le modulo per

- `import modulo`; post que nos voca functiones per nomines `modulo.f1`, `modulo.f2`, et cetera.
- `import modulo as md`; post que nos voca functiones per nomines `md.f1`, `md.f2`, et cetera.
- `from modulo import (f1,f2)`; post que nos pote usar le functiones listate (hic `f1` e `f2`) sin le nomine de modulo.
- `from modulo import *`; post que nos pote usar omnes le functiones de modulo sin le nomine de modulo.

In modulo `math` nos trova functiones e structuras mathematic, per exemplo, le constantes `e` et `pi`.

Ecce le prime forma a usar nomines:

```
import math
> math.e
2.718281828459045
> math.pi
3.141592653589793
```

Hic un altere forma:

```
> from math import *
> e
2.718281828459045
> pi
3.141592653589793
```

Capitulo 2

Programma de numerales

In iste capitulo nos continua apprender le linguage Python per le exemplos de numerales de Interlingua.

Le function `print`

Nos usa le function `print` a monstrar sequentias de characteres sur terminal texto.

Le function `print` pote haber argumentos de differente typos. Le interprete adde spatios inter cata argumento.

```
> print ("Un die ha",24*60*60,"secundas.")
Un die ha 86400 secundas.
```

Il es possibile da le clave `end` como argumento a function `print`. Le clave `end` es un sequentia de characteres addite in fin de texto.

Normalmente le function `print` adde in fin de texto un signo a cambiar le linea (`"\n"`). Ecce exemplos a usar le clave `end`:

```
> print (20, end=" "); print (20)
20 20
> print (20, end=""); print (20)
2020
```

Le operator `in`

Le expression `(x in xs)` es `True`, quando `x` es un elemento de `xs`, e `False` alteremente.

```
> vocales = "eaiouy"
> "a" in vocales
True
> "t" in vocales
False
```

Le instruction `for`

Le instruction `for` transverse un sequentia iterabile:

```
> for c in "eaiou":
|   print (c, end=' ')
|
e a i o u
```

Le instruction `if`

Le instruction `if` ha le forma

```
if expression_1:
  functiones_1
elif expression_2:
  functiones_2
else:
  functiones_3
```

ubi `functiones_1` es vocate quando `expression_1` es considerate ver. In altere caso le `functiones_2` de parte `elif` es vocate quando `expression_2` es ver, et cetera. Si cata expression es considerate false, le `functiones_3` de parte `else` es vocate. Le nomine `elif` veni ex le parolas anglese *else if*.

Un expression es considerate false quando il ha le valor `None`, `False`, `0`, `0.0`, `''`, `()`, `[]`, `{}`, et cetera. In altere casos le expression es considerate ver.

Comprehensiones

Un methodo a formar un lista de iterabiles, es usar *comprehensiones*. Comprehensiones es expressiones inter parentheses quadrate `[…]` con un o plure instructiones `for`, possibilemente con instructiones `if`.

```
> vocales = "eaiouy"
> [c for c in vocales]
['e', 'a', 'i', 'o', 'u', 'y']

> alfabeto = "eaionlsrtucdmpvbghfqxjykzw"
> consonantes = [c for c in alfabeto if c not in vocales]
> consonantes
['n', 'l', 's', 'r', 't', 'c', 'd', 'm', 'p', 'v', 'b', 'g', 'h',
 'f', 'q', 'x', 'j', 'k', 'z', 'w']
```

Generatores

Generatores es functiones que genera valores a *iterar*, id es, valores que nos pote transversar.

Le function `range` es un generator simple, que genera a nos sequentias arithmetic. Nos trova le forma de vocar de function `range` quando nos scribe `help (range)` al interprete:

```
> help (range)
class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
...
```

Le valor de function `range` es talmente un objecto de `range`. Le argumento `start` es le comencio de sequentia, le argumento `stop` le fin. Le comencio es parte de sequentia, ma le fin non es: `(start <= x < stop)`.

- Le forma breve `range (stop)` da nos un objecto `range` de elementos de `0` a `(stop - 1)`. Le amonta de elementos es `stop`.
- Le forma `range (start,stop)` da nos un objecto `range` de elementos de `start` a `(stop - 1)`. Le amonta de elementos es `(stop - start)`.

- Le forma `range (start,stop,step)` da nos un objecto `range` de elementos de `start` a `(stop - 1)` con le intervallo `step`.

Le comprehensiones da nos un medio natural a representar elementos generate per `range`:

```
> [x for x in range (5)]
[0, 1, 2, 3, 4]
> [x for x in range (3,7)]
[3, 4, 5, 6]
> [x for x in range (1,11,2)]
[1, 3, 5, 7, 9]
> [x for x in range (5,-1,-1)]
[5, 4, 3, 2, 1, 0]
```

Nomines de numerales: unes

In le lista `unes` nos ha definite le nomines de numeros 1…9.

```
unes = [ '', 'un', 'duo', 'tres', 'quatro', 'cinque',
 'sex', 'septe', 'octo', 'nove' ]
```

Nos memora que le indice de un lista comencia per zero, e pro isto nos defini le prime elemento de nostre lista a un valor vacue (`''`). Nunc le altere numeros remane in su loco correcte:

```
> unes [0]
''
> unes [3]
'tres'
```

Le unes de un numero es sempre le residuo de division per 10. Per exemplo,

```
> 5764 % 10
4
```

Si le integre numero `n` es 0, le resultato es "zero". Alteremente il existe parola pro unes solmente si `(n % 10 > 0)`. Nos defini le function `unes1`, que ha le argumento `n` (le numero integre).

```
def unes1 (n):
```

```
  i = n % 10
  result = ''
  if n == 0:
    result = 'zero'
  elif i > 0:
    result = unes [i]
  return result
```

Si le numero integre es 3456, nos ha ”sex” unes. Si le numero es 40, nos ha 0 unes, e nos non vole dicer ”quaranta-zero”. Solmente si le integre numero es 0, e nos ha 0 unes, le parola es ”zero”.

```
> unes1 (3456)
'sex'
> unes1 (40)
''
> unes1 (0)
'zero'
```

Deces in un numeral

Le nomines a deces nos ha in lista `deces`.

```
deces = [ '', 'dece', 'vinti', 'trenta', 'quaranta', 'cinquanta',
  'sexanta', 'septanta', 'octanta', 'novanta' ]
```

Le numero 3456 ha `(n % 100 // 10)`, id es, 5 deces, e talmente le parola es ”cinquanta”.

```
> deces [5]
'cinquanta'
```

Centos in un numeral

Le numero integre `n` sempre ha `(n % 1000 // 100)` centos. Per exemplo,

```
> 3456 % 1000 // 100
4
> 12 % 1000 // 100
```

```
0
```

Quando nos ha 1 de centos, le parola pro isto es ”cento”. Quando nos ha plure de centos, nos adjunge le numeral de unes (”duo”, ”tres”, ”quatro”, …) al forma plural ”centos”. Nos anque debe haber un spatio inter le parolas. Le function `centos` deveni

```
def centos (n):
  c = n % 1000 // 100
  result = ''
  if c == 1:
    result = 'cento'
  if c > 1:
    result = unes [c] + ' centos'
  return result
```

Nunc per exemplo,

```
> centos (100)
'cento'
> centos (200)
'duo centos'
```

Le union de centos, deces, e unes

Nos nunc ha le centos, le deces, e le unes de numero `n`:

```
c = centos (n)
d = deces [n % 100 // 10]
u = unes1 (n)
```

Quando nos adjunge le parolas, nos debe notar, que pote mancar alicun de parolas, atque etiam omnes de illos. Pro isto nos defini function `join_numwds`:

```
def join_numwds (a,b,sep=" "):
  result = ''
  if a and b:
    result = a + sep + b
  elif a:
    result = a
```

```
  elif b:
    result = b
  return result
```

Le function `join_numwds` ha un clave `sep` como argumento. Isto naturalmente es le *separator* inter parolas. Quando, per exemplo, le numero es "cento un", le separator es un spatio `' '`. Quando le numero es "dece-duo" le separator es un linea `'-'`. Usualmente le separator es un spatio, e pro isto argumento `sep` deveni le argumento predefinite con le valor `' '`. Isto significa que nos pote omitter le argumento quando illo ha le valor predefinite.

Le expression `(a and b)` deveni ver, quando et `a` et `b` ha un valor differente a un sequentia vacue `''`. Tunc nos adjunge sequentias `a`, `sep` et `b`.

Alteremente le resultato es `a` quando `a` existe, o `b` quando `b` existe. Si necuno existe, le resultato es un sequentia vacue `''`.

Alora, nos adjunge deces e unes per un linea `'-'` e le centos con le resultato de iste per un spatio `' '`. Nos memora que un spatio es argumento predefinite, e talmente nos pote omitter lo.

```
def from1to999 (n):
  c = centos (n)
  d = deces [n % 100 // 10]
  u = unes1 (n)
  w1 = join_numwds (d,u,"-")
  w2 = join_numwds (c,w1)
  return w2
```

Milles

Quando nos calcula le milles de un numero, nos pote utilisar le function pro centos, deces e unes. Le numero `n` sempre ha `(n % 1_000_000 // 1_000)` milles. Per exemplo le numero 1000 ha un mille, le 5000 ha cinque milles e 123 000 ha cento vinti-tres milles. Le function pro 1, 5, e 123 es le mesme function `from1to999` que nos usava pro unes.

Le function `milles` nunc es

```
def milles (n):
```

```
  m = n % 1_000_000 // 1_000
  result = ''
  if m == 1:
    result = 'mille'
  if m > 1:
    result = from1to999 (m) + ' milles'
  return result
```

Hic nos vide que in Python nos pote gruppar le numeros con un tracto de sublineamento _, per exemplo un million deveni `1_000_000`.

Le numero integre

Nos nunc ha omnes le partes pro calcular numeros de 0 a 999 999:

```
def numeral (n):
  m = milles (n)
  c = from1to999 (n)
  w = join_numwds (m,c)
  return w
```

Que nos testa le algorithmo per numeros aleatori!

Le modulo `random`

Le modulo `random` produce numeros aleatori.

Per exemplo le function `randint` da nos un numero integre inter duo numeros. Le function `random` da nos un numero de typo `float` inter 0 e 1.

Le function `choice` selige un elemento aleatori ex un lista. Le function `shuffle` misce le elementos de un lista.

```
> random.randint (1,3)
3
> random.random ()
0.8974826518988037
> random.choice ([3,5,7,9])
7
```

```
> t = ['A','B','C']
> random.shuffle (t)
> t
['C', 'A', 'B']
```

Numerales aleatori in parolas

Le test de numerales aleatori deveni

```
for i in range (1,5):
  n = random.randint (0,20_000)
  print (n, numeral (n))
```

Le resultatos es

```
16631 dece-sex milles sex centos trenta-un
7650 septe milles sex centos cinquanta
4800 quatro milles octo centos
117 cento dece-septe
```

Methodo `str.join`

Sequentias de characteres ha le methodo `join`, que adjunge le elementos de su argumento con le instantia de classe `str`. Per exemplo, `" ".join (["A","B","C"])` adjunge le elementos `"A"`, `"B"` et `"C"` con le sequentia `" "`.

```
> " ".join (["A","B","C"])
'A B C'
```

Quando nos vole colliger numeros ex un sequentia de characteres, nos pote utilisar comprehensiones e methodo `join`:

```
> s = "5i739iA39j55"
> "".join ([c for c in s if c in "1234567890"])
'57393955'
```

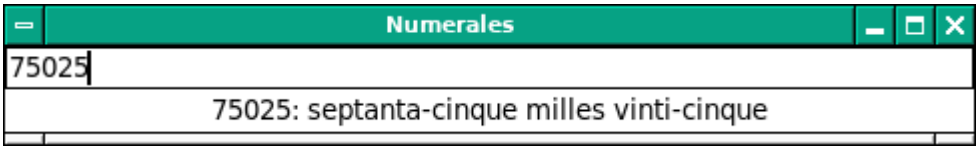

Figura 2. Nostre programma "Numerales".

Secunde programma graphic

Nos ha usate le componente `Label` a monstrar un texto in un fenestra. Nunc nos adde un componente `Entry` a leger un sequentia de characteres.

Ambes ha un lista de proprietates que nos pote usar a cambiar le aspecto de elemento: su parente, color, bordo, largessa, typo de litteras, et cetera. Nos selige nulle bordo, texto nigre sur fundo blanc. Nomines de numerales es longe: nos selige largessa 60 characteres (figura 2).

Quando le fenestra appare, nos da le focus al componente `Entry`, post que nos pote scriber in fenestra sin cliccar le componente.

```
root = Tk ()
root.title ('Numerales')
sv = StringVar ()
e = Entry (root, textvariable=sv, bd=0, width=60)
w = Label (root, bg="white", fg="black", width=60)
e.pack ()
w.pack ()
sv.trace ("w", lambda name, index, mode, sv=sv: callback (sv,w))
e.focus_set ()
root.mainloop ()
```

Hic nos ha prestate un function `callback`, que responde al cambios de texto. Le texto es in variabile `sv` de classe `StringVar`. Nos lege le texto per methodo `get`.

Nos sape como colliger le numeros ex texto. Quando nos scribe solmente le numeros, nos ha validate le entrata.

```
def callback (sv,w):
  s = sv.get ()
```

```
  n = "".join ([c for c in s if c in "1234567890"])
  sv.set (n)
  numero = int_def (n)
  if numero < 1_000_000:
    result = numeral (numero)
  else:
    result = 'troppo'
  t = str (int_def (n)) + ": " + result
  print (t)
  w ['text'] = t
```

Le function `int` retorna le numero integre de un sequentia de characteres (per exemplo, `int ("12") == 12`). Si le contento de sequentia non es un numero, le resultato es un error (plus exacte un `ValueError`).

Nos defini un function `int_def` que retorna un valor predefinite per clave `default` si le sequentia de characteres non es un numero. Un tal caso eveni in nostre programma quando le sequentia es vacue `''`.

```
def int_def (st,default=0):
  try:
    result = int (st)
  except ValueError:
    result = default
  return result
```

Capitulo 3

Classes

Classes da nos un instrumento a unir le datos e le functionalitate. Quando nos crea un nove classe, nos crea un nove *typo* de objecto, e nos pote crear nove *instantias* de iste typo. Cata instantia de un classe pote haber *attributos*, id es, characteristicas proprie a un objecto, que mantene le *stato* del objecto. Instantias de classe pote anque haber *methodos* pro modificar le stato del objecto.

Le classe minimal

Nostre prime classe a definir es un classe minimal, `A`, a que nos non defini ni methodos ni attributos:

```
class A:
  pass
```

Le instruction `pass` dice a facer nihil. — Isto es un classe minimal.

Nos defini un instantia `a` de classe `A`.

```
> a = A ()
```

Nos pote definir le nomines `a.x` et `a.y`:

```
> a.x = 4
> a.y = 1
```

```
> a.x, a.y
(4, 1)
```

Le attributos e methodos

Quando nos crea objectos, nos defini le valores initial al attributos de objectos in methodo `__init__`.

```
class B:
  def __init__ (self, x, y):
    self.x = x
    self.y = y
```

Cata definition de un methodo ha `self` como prime argumento. Isto es un formalitate a facer le objecto mesme usabile a methodo.

Nos crea un objecto `b` de classe `B`, con initial valores `x = 3` et `y = 4`:

```
> b = B (3,4)
```

In le definition de classe, nos defini le methodo `methodo_x`. Le definition ha al minus un argumento, `self`, le objecto mesme:

```
  ...
  def methodo_x (self):
    pass
  ...
```

Nos voca le methodo. Nunc le argumento `self` ha le valor `self = b`.

```
b.methodo_x ()
```

Si nos vole, nos pote initiar `x` et `y`, per exemplo, a valores `x = 0` et `y = 0` (puncto origine).

```
class C:
  def __init__ (self, x=0, y=0):
    self.x = x
    self.y = y
```

Nos defini un instantia `c` de classe `C`. Le instantia nunc ha le attributos `x` et `y`, con valores `c.x = 0` et `c.y = 0`.

```
> c = C ()
> c.x, c.y
(0, 0)
```

Le classe `Point`

Nos nunc defini un classe `Point` que es simile a classes `A`, `B`, et `C` supra, ma que ha anque altere methodos:

```
class Point:
  def __init__ (self, x=0, y=0):
    self.x = x
    self.y = y
  def __repr__ (self):
    return f"Point ({self.x},{self.y})"
  def pair (self):
    return (self.x,self.y)
```

Le methodo `__repr__` es le *representation* de un objecto de classe in question. Nos vide iste representation quando nos scribe le nomine de un objecto in interprete interactive:

```
> p = Point ()
> p
Point (0,0)
```

Nos anque definiva un methodo `pair`, que retorna a nos un 2-tupla con le coordinatas `(x,y)`.

```
> p.pair ()
(0, 0)
```

Le classes `Line` e `Polygon`

Nos defini un objecto de classe `Line` per duo punctos `p1` et `p2`. Un objecto de class `Polygon` ha un attributo `ps` que es un lista de punctos.

```
class Line:
  def __init__ (self, p1, p2):
```

```
    self.p1 = p1
    self.p2 = p2

class Polygon:
  def __init__ (self, ps):
    self.ps = ps
```

Le f-formato de sequentias de characteres

Le *f-formato* es un maniera de formation de sequentias de characteres.

Nos scribe le littera `f` o `F` ante le prime virguletta, e le expressiones inter parentheses crispe `{…}`. Il ha differente manieras a manipular le aspecto de sequentia, per exemplo `f"{x:.2f}"` da le valor de `x` con 2 decimales.

```
> twopi = 6.2831853
> f"Le perimetro de circulo unitate es circa {twopi:.2f}."
'Le perimetro de circulo unitate es circa 6.28.'
> a,b = 6,7
> f"{a} vices {b} es {a*b}."
'6 vices 7 es 42.'
```

Modulo `cairo`

Nos usa le modulo `cairo` a pinger picturas. Iste vice nos face un pictura de formato PDF. Prime nos crea un contexto `ct` a pinger. Nos face le area vacue quando nos pinge un rectangulo blanc de puncto $(0, 0)$ al angulo opposite de pictura.

Le modulo `cairo` ha le functiones a pinger rectangulos, a eliger un color, e a plenar un forma con le color. Le functiones es le methodos `rectangle`, `set_source_rgb`, e `fill` de instantias de classe `cairo.Context`.

```
w_pic, h_pic = 170,130
w,h = 160,120
filename = "armature-1.pdf"
surface = cairo.PDFSurface (filename, w_pic, h_pic)
ct = cairo.Context (surface)
```

```
ct.rectangle (0, 0, w_pic, h_pic)
ct.set_source_rgb (1.00, 1.00, 1.00)
ct.fill()
```

Nos defini functiones `draw_rect`, `draw_line` e `draw_polygon` a pinger un rectangulo, un linea e un polygono. Nos nunc besonia le methodos `move_to`, `line_to` e `close_path` de nostre objecto `ct`.

```
def draw_rect (p1,p2):
  ct.rectangle (p1.x, p1.y, p2.x - p1.x, p2.y - p1.y)

def draw_line (line):
  p1,p2 = line.p1,line.p2
  ct.move_to (p1.x,p1.y)
  ct.line_to (p2.x,p2.y)

def draw_polygon (ps):
  ct.move_to (ps[0].x,ps[0].y)
  for t in ps[1:]:
    ct.line_to (t.x,t.y)
  ct.close_path ()
  ct.set_source_rgb (1.00, 0.50, 0.00)
  ct.fill ()
```

Nos nunc pinge le pictura (figura 3):

```
x1,y1 = 5,5
x2,y2 = x1 + w, y1 + h

pairs1 = [(x1,y1),(x2,y1),(x2,y2),(x1,y2)]
ps = [Point (a,b) for (a,b) in pairs1]

draw_line (Line (ps[0],ps[2]))
draw_rect (ps[0],ps[2])

ct.set_source_rgb (0.00, 0.00, 0.00)
ct.set_line_width (1.0)
ct.stroke ()
```

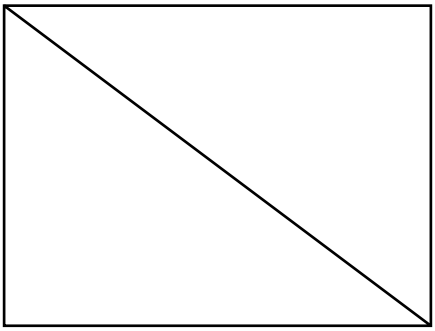

Figura 3. Pictura con un rectangulo e un diagonal.

Intersection de duo lineas

Nos defini un function `intersection` que calcula le puncto de intersection inter duo lineas.

```
def intersection (line1,line2):
  """Line-Line intersection

  From: en.wikipedia.org/wiki/Line-line_intersection
  """
  p1,p2 = line1.p1,line1.p2
  p3,p4 = line2.p1,line2.p2
  (x1,y1),(x2,y2) = (p1.x,p1.y),(p2.x,p2.y)
  (x3,y3),(x4,y4) = (p3.x,p3.y),(p4.x,p4.y)
  nx = (x1*y2-y1*x2) * (x3-x4) - (x1-x2) * (x3*y4-y3*x4)
  ny = (x1*y2-y1*x2) * (y3-y4) - (y1-y2) * (x3*y4-y3*x4)
  d  = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4)
  parallel = (d == 0)
  if parallel:
    return None
  else:
    x  = nx / d
    y  = ny / d
    return Point (x,y)
```

Hic nos trova un maniera a scriber sequentias de characteres que contine saltos de lineas: le virgulettas triple, `'''…'''` o `"""…"""`. In le prime linea de un definition illos forma un documentation pro definition, que nos vide scribente `help (intersection)` e que es situate in attributo `intersection.__doc__`.

```
> intersection.__doc__
'Line-Line intersection\n\n  From: en.wikipedia.org/...\n  '
```

Nos anque defini un function `draw_point` a pinger un puncto.

```
def draw_point (pt):
  ct.arc (pt.x,pt.y,3,0,math.tau)
  ct.set_source_rgb (0, 0, 0)
  ct.fill ()
```

Nos pinge le diagonales del rectangulo, e calcula lor puncto de intersection.

```
diagonal1 = Line (ps[0],ps[2])
diagonal2 = Line (ps[1],ps[3])

draw_line (diagonal1)
draw_line (diagonal2)

c1 = intersection (diagonal1,diagonal2)
```

Finalmente nos pinge le punctos (figura 4). Hic `ps` es un lista de punctos, e nos pote usar le operator `+` a unir duo listas, quando le ambe operandos es listas: puncto `c1` deveni lista per parentheses quadrate `[c1]`.

```
for p in ps + [c1]:
  draw_point (p)
```

Operator + inter duo objectos

Similarmente que nos definiva methodo `__repr__` a un objecto, nos pote definir le operator `+` a adder duo objectos. Le methodo a definir es `__add__`. In `a + b` le prime operando `a` a adder deveni le argumento `self`. Le secunde operando `b` deveni le argumento secunde: `__add__ (self, b)`.

Hic nos defini addition inter duo punctos, id es, inter duo objectos de classe

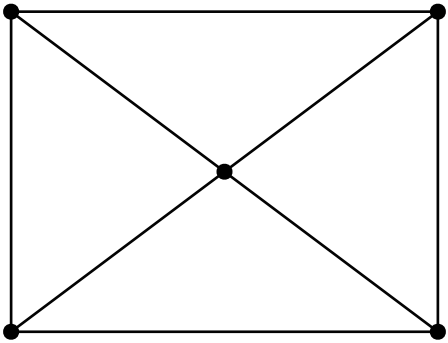

Figura 4. Diagonales de un rectangulo e punctos de intersection.

`Point`:

```
class Point:
  ...
  def __add__ (self, p2):
    return Point (self.x + p2.x, self.y + p2.y)
  ...
```

Nunc le addition de duo objectos es le addition de su vectores de position.

```
> a = Point (1,2)
> b = Point (3,4)
> a + b
Point (4,6)
```

Le centro inter duo punctos es

```
def center (a,b):
  return Point ((a.x + b.x) / 2, (a.y + b.y) / 2)
```

Le function `enumerate`

Le function `enumerate` es un generator que numera un sequentia:

```
> list (enumerate ("eaiou"))
[(0, 'e'), (1, 'a'), (2, 'i'), (3, 'o'), (4, 'u')]
```

Usante un numeration, nos trova le indices de un lista, e nos pote facer calculationes per illos. Per exemplo, un circlo al transverso de indices es

```
cs = [center (ps[i],ps[(i+1)%len(ps)]) for i,e in enumerate (ps)]
```

Si le longor de lista `ps` es 6, le function `enumerate` da al variabile `i` le valores 0…5. Nunc quando `i = 5` le indice `(i+1)` es $5 + 1 = 6$ e $6 \bmod 6 = 0$, e nos ha le ultime elemento `(ps[5],ps[0])` que completa le circulo.

Insimules

In Python un *insimul* (anglese: *set*) adhere le notation de parentheses crispe `{…}` de un dictionario. Le elementos de un insimul tamen es valores, e non de forma *clave*:*valor*. Le `set ()` es un insimul vacue. (Le `{}` es un dictionario vacue.)

```
> type ({})
<class 'dict'>
> type (set ())
<class 'set'>
```

Nos usa insimules quando nos vole que cata elemento del insimul es unic.

```
> {1,2,1,3,2,4}
{1, 2, 3, 4}
```

Lineas inter cata puncto

Le elementos de lista `cs` se trova inter elementos de lista `ps`. Nos prende un elemento de ambe lista, e extende le nove lista `ts` correspondente.

```
ts = []
for (a,b) in zip (ps,cs):
  ts.extend ([a,b])
```

Quando nos trenchava le lineas per centros, alicun de illes veniva superflue:

```
cutd = [(0,2),(2,4),(4,6),(0,6)]
```

Nos nunc ha un bon comprehension in forma de un insimul `s1`:

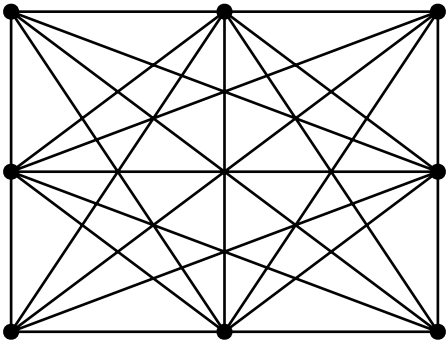

Figura 5. Lineas inter punctos.

```
t = range (len (ts))
s1 = {(a,b) for a in t for b in t if a < b if (a,b) not in cutd}
ks = [Line (ts[a],ts[b]) for (a,b) in s1]
```

Quando nos pinge le lineas e le punctos, nos trova un nove pictura (figura 5).

```
for k in ks:
  draw_line (k)
ct.stroke ()

for p in ts:
  draw_point (p)
```

Modulo `subprocess`

Nos pote monstrar le pictura per un visualisator externe. Hic nos usa le programma `mupdf`. Le modulo `subprocess` ha function `Popen` a initiar un processo externe.

```
import subprocess

filename = "armature-4.pdf"
subprocess.Popen (["mupdf " + filename],shell=True)
```

Capitulo 4

Dictionarios de files texto

In capitulo 2 nos preparava functiones a transformar numeros a sequentias de characteres. Nos definiva listas como `unes` e `deces`, e functiones como `milles`, `centos`, `unes1`, et cetera. Omnes le functiones nos salvava in file `numerales.py`.

Nos nunc pote utilisar file `numerales.py` in le interprete de Python. Ex functiones le function `numeral` es le plus utile, le alteres nos hodie non besonia. Si le file `numerales.py` es in le mesme directory ubi nos initia le interprete, nos pote importar lo como un modulo:

```
> from numerales import numeral
> numeral (1994)
'mille nove centos novanta-quatro'
```

Nos construe le prime 50 milles numeros e salva los in file `numeros.txt`:

```
lines = []
for i in range (0,50_000):
  lines.append (f"{i}: {numeral(i)}\n")
with open ("numeros.txt","w") as f:
  f.writelines (lines)
```

Le file `numeros.txt` nunc ha 50 000 lineas de numeros:

```
0: zero
```

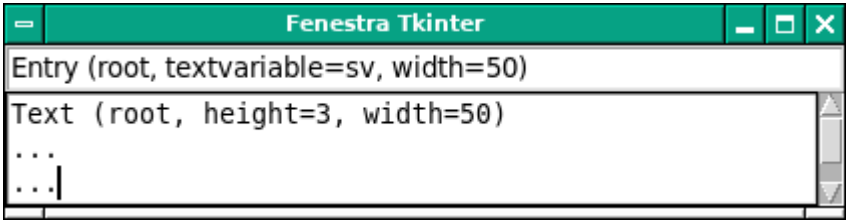

Figura 6. Componente Text.

```
1: un
2: duo
3: tres
...
49999: quaranta-nove milles nove centos novanta-nove
```

Le componente `Text`

Le componente `Text` in modulo `tkinter` es un campo de texto de plure lineas. Nos crea illo como le altere componentes in modulo `tkinter` (figura 6).

```
from tkinter import *

root = Tk ()
root.title ('Fenestra Tkinter')

sv = StringVar ()
en = Entry (root, textvariable=sv, width=50)
sc = Scrollbar (root)
te = Text (root, height=3, width=50)

en.pack (side=TOP, fill=X)
sc.pack (side=RIGHT, fill=Y)
te.pack (side=LEFT, fill=Y)

sc.config (command=te.yview)
te.config (yscrollcommand=sc.set)
```

```
sv.set ("Entry (root, textvariable=sv, width=50)")
txt = "Text (root, height=3, width=50)\n...\n...\n"
te.insert (END,txt)

root.mainloop()
```

Nos da le focus al componente `en`, que es de typo `Entry`. Quando nos pressa le clavo `<Return>`, nos voca le function `return_pressed`.

```
en.bind('<Return>', return_pressed)
en.focus()
```

Nos defini le function `return_pressed` a vocar le function `find_words`, Nos omitte le argumento `event`.

```
def return_pressed(event):
  find_words()
```

Le function `find_words` cerca le texto de campo `en` (le componente de typo `Entry`) inter le lineas, e da nos al maximo `max_m = 200` lineas como resultato. Le lineas de resultato nos inserta in le componente `te` (que esseva le componente de typo `Text`). Le componente de typo `Text` es un campo de texto de plure lineas, e il existe differente formas a exprimer su indice, per exemplo `"linea.columna"` o `CURRENT` o `END`. Le indice `"1.0"` es le prime columna de prime linea.

```
def find_words():
  max_m = 200
  start = time.time()
  word = en.get()
  result = []
  if len(word) > 0:
    m = 0
    i = 0
    while m <= max_m and i < len (lines):
      s = lines [i]
      i = i + 1
      if word in s:
        result.append (s)
        m = m + 1
```

```
  end = time.time()
  print(end - start)
  en.delete(0, END)
  te.delete("1.0", END)
  te.insert("1.0","".join(result))
  if (m > max_m):
    te.insert(END,"...")
```

Quando nos initia le programma, Python ha le argumentos in lista `sys.argv`. Le prime elemento `sys.argv[0]` es sempre le nomine de programma. Nos usa le altere argumentos a leger le textos in lista `lines`.

```
import sys

lines = []
for i in range (1, n):
  filename = sys.argv[i]
  with open (filename) as f:
    lines = lines + f.readlines()
```