Pyntro

TiN

24 de Abril del 2010

Algunas Características

- LIBRE

- Fácil de aprender.
  Se lee como pseudo-código.
  Sintaxis sencilla, lenguaje muy ortogonal.

Algunas Características

- LIBRE
- Maduro (19 años).
  Diseño elegante y robusto, en constante evolución
- Fácil de aprender.
  Se lee como pseudo-código.
  Sintaxis sencilla, lenguaje muy ortogonal.

Algunas Características

- LIBRE
- Maduro (19 años).
  Diseño elegante y robusto, en constante evolución
- Fácil de aprender.
  Se lee como pseudo-código.
  Sintaxis sencilla, lenguaje muy ortogonal.
- Extremadamente portable.
  Unix, Windows, Mac, BeOS, Win/CE
  DOS, OS/2, Amiga, VMS, Cray...

Propiedades del Lenguaje

- Compila a bytecode interpretado.
  La compilación es implícita y automática.
  Tipado dinámico, pero fuerte.

Propiedades del Lenguaje

- Compila a bytecode interpretado.
  La compilación es implícita y automática.
  Tipado dinámico, pero fuerte.
- Multi-paradigma.
  Todo son objetos.
  Pero puede usarse de manera procedural.

Propiedades del Lenguaje

- Compila a bytecode interpretado.
  La compilación es implícita y automática.
  Tipado dinámico, pero fuerte.
- Multi-paradigma.
  Todo son objetos.
  Pero puede usarse de manera procedural.
- Módulos, clases, funciones, generadores.

Propiedades del Lenguaje

- Compila a bytecode interpretado.
  La compilación es implícita y automática.
  Tipado dinámico, pero fuerte.
- Multi-paradigma.
  Todo son objetos.
  Pero puede usarse de manera procedural.
- Módulos, clases, funciones, generadores.
- Viene con las baterias incluidas.
  Extensa biblioteca estándar.
  Clave en la productividad de Python.

Más Propiedades...

- Manejo moderno de errores.
  Por excepciones.
  Muy útil detalle de error.

Más Propiedades...

- Manejo moderno de errores.
  Por excepciones.
  Muy útil detalle de error.
- Tipos de datos de alto nivel.
  Enteros sin límites, strings, flotantes, complejos.
  Listas, diccionarios, conjuntos.

Más Propiedades...

- Manejo moderno de errores.
  Por excepciones.
  Muy útil detalle de error.
- Tipos de datos de alto nivel.
  Enteros sin límites, strings, flotantes, complejos.
  Listas, diccionarios, conjuntos.
- Módulos, clases, funciones, generadores.

Más Propiedades...

- Manejo moderno de errores.
  Por excepciones.
  Muy útil detalle de error.
- Tipos de datos de alto nivel.
  Enteros sin límites, strings, flotantes, complejos.
  Listas, diccionarios, conjuntos.
- Módulos, clases, funciones, generadores.
- Intérprete interactivo.
  Clave en el bajo conteo de bugs.
  Acelera sorprendentemente el tiempo de desarrollo.
  Permite explorar, probar e incluso ver la documentación.

La Biblioteca estándar ayuda con...

Servicios del sistema, fecha y hora, subprocesos, sockets, internacionalización y localización, base de datos, threads, formatos zip, bzip2, gzip, tar, expresiones regulares, XML (DOM y SAX), Unicode, SGML, HTML, XHTML, XML-RPC (cliente y servidor), email, manejo asincrónico de sockets, clientes HTTP, FTP, SMTP, NNTP, POP3, IMAP4, servidores HTTP, SMTP, herramientas MIME, interfaz con el garbage collector, serializador y deserializador de objetos, debugger, profiler, random, curses, logging, compilador, decompilador, CSV, análisis lexicográfico, interfaz gráfica incorporada, matemática real y compleja, criptografía (MD5 y SHA), introspección, unit testing, doc testing, etc., etc...

Python Argentina

- Grupo de entusiastas de Python.
  Referencia para la aplicación y difusión del lenguaje.
- ¿Cómo participar?
  Suscribiéndose a la Lista de Correo (somos +700).
  Más info en la página: `http://www.python.org.ar`

Corriendo e interpretando

Menos charla y mas acción

- Python es interpretado.
- No hace falta compilar.
- Ciclo corto de pruebas.
- Y encima tenemos el Intérprete Interactivo.

Tipos de Datos

- Haciendo números, y más números.
- Cadenas, y como accederlas.
- Listas, listas, y muchas listas.
- Conjuntos.
- Diccionarios, ¡diccionarios!

Haciendo Números

Enteros

```
>>> 4 / 2
2
>>> 4 %2
0
>>> 92345678765432345678 98 * 9876543223456787654
912056087826248206341022710030958311312 92
```

Floats

```
>>> 3 * 3.75 / 1.5
7.5
>>> 7 / 2.3
3.043478260869565 4
```

Más Números

Complejos

```
>>> 2 + 3j
(2+3j)
>>> (2+3j * 17) ** (2+5j)
(-0.9125883266746936-0.8249833362981151 6j)
```

Recortando los decimales

```
>>> int(12.3)
12
>>> round(2.7526)
3.0
>>> round(2.7526, 2)
2.75
```

Cadenas

Comillas, apóstrofos, triples

```
>>> 'Una cadena es una secuencia de caracteres'
'Una cadena es una secuencia de caracteres'
>>> """Una linea
... y la otra"""
'Una linea\ny la otra'
```

Algunas operaciones

```
>>> "Hola" + " mundo"
'Hola mundo'
>>> "Eco " * 4
'Eco Eco Eco Eco '
>>> len("Hola mundo")
10
```

Accediendo a las cadenas

Por posición

```
>>> saludo = 'Hola mundo'
>>> saludo[0]
'H'
>>> saludo[3]
'a'
```

Rebanando

```
>>> saludo[2:5]
'la '
>>> saludo[2:8]
'la mun'
>>> saludo[-2:]
'do'
```

Listas

Corchetes, varios tipos de elementos

```
>>> a = ['harina', 100, 'huevos', 'manteca']
>>> a
['harina', 100, 'huevos', 'manteca']
```

Accedemos como cualquier secuencia

```
>>> a[0]
'harina'
>>> a[-2:]
['huevos', 'manteca']
```

Listas

Concatenamos, reemplazamos

```
>>> a + ['oro', 9]
['harina', 100, 'huevos', 'manteca', 'oro', 9]
>>> a[0] = "sal"
>>> a
['sal', 100, 'huevos', 'manteca']
```

Pueden tener incluso otras listas

```
>>> a
['sal', 100, 'huevos', 'manteca']
>>> a[1] = ["Hola", 7]
>>> a
['sal', ['Hola', 7], 'huevos', 'manteca']
```

Y dale con las listas

Borramos elementos

```
>>> del a[-1]
>>> a
['sal', ['Hola', 7], 'huevos']
```

Tenemos otros métodos

```
>>> a.index("huevos")
2
>>> a.sort()
>>> a
[['Hola', 7], 'huevos', 'sal']
```

Conjuntos

Definimos con set()

```
>>> juego = set("juego")
>>> juego
set(['e', 'j', 'u', 'o', 'g'])
>>> hechizo = set("echo")
>>> hechizo.update(set("izo"))
>>> hechizo
set(['c', 'e', 'i', 'h', 'o', 'z'])
```

Conjuntos

Operamos

```
>>> juego - hechizo
set(['j', 'u', 'g'])
>>> hechizo & juego
set(['e', 'o'])
>>> hechizo.remove("h")
>>> hechizo.add("Merlin")
>>> hechizo
set(['c', 'Merlin', 'e', 'i', 'o', 'z'])
```

Diccionarios

Definimos con llaves

```
>>> dias = {"enero": 31, "junio": 30, "julio": 30}
>>> dias
{'julio': 30, 'enero': 31, 'junio': 30}
>>> dias["enero"]
31
>>> dias["agosto"] = 31
>>> dias["julio"] = 31
>>> dias
{'julio': 31, 'enero': 31, 'junio': 30, 'agosto': 31}
```

Diccionarios

Borrando

```
>>> del dias["julio"]
>>> dias
{'enero': 31, 'junio': 30, 'agosto': 31}
```

Viendo qué hay

```
>>> "marzo" in dias
False
>>> dias.keys()
['enero', 'junio', 'agosto']
>>> dias.values()
[31, 30, 31]
```

Más Diccionarios

Otros Métodos

```
>>> dias.get("agosto", "No tenemos ese mes")
31
>>> dias.get("mayo", "No tenemos ese mes")
'No tenemos ese mes'
>>> dias.pop("agosto")
31
>>> dias
{'enero': 31, 'junio': 30}
```

Control de Flujo

Estructura del if

```
a = ...
if a == 0:
    print "Ojo con el valor de b"
    b = 0
elif a > 100 or a < 0:
    print "Error en el valor de a"
    b = 0
print b
```

Eso que hay después del if:

- or, and, not
- < > == != in is
- Todo evalúa a Falso o Verdadero

Por cada uno

Estructura del for

```
>>> bichos = ["pulgas", "piojos"]
>>> for bicho in bichos:
...         print "Mata-" + bicho
Mata-pulgas
Mata-piojos
```

Si queremos la secuéncia de números

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> for i in range(5):
...         print i**2
0 1 4 9 16
```

Mientras Tanto

Estructura del while

```
>>> a = 0
>>> while a<1000:
...         print a**5
...         a += 3
0
243
...
995009990004999
```

Al igual que el for, tiene:

- Continue: Vuelve a empezar el loop
- break: Corta el loop y sale
- else: Lo ejecuta, si no cortamos en el break

Excepciones

Suceden cuando algo se escapa de lo normal

```
>>> 14 / 2
7
>>> 14 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

Podemos capturarlas

```
>>> try:
...     print 14 / 0
... except ZeroDivisionError:
...     print "error!"
error!
```

Manejando lo excepcional

Es muy versátil

- try: Acá va el bloque de código que queremos supervisar
- except: Atrapa todo, o sólo lo que se le especifique
- else: Si no hubo una excepción, se ejecuta esto
- finally: Lo que esta acá se ejecuta siempre

Se pueden combinar de cualquier manera

Y también podemos generar excepciones

```
>>> raise ValueError("Aca contamos que pasó")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: Aca contamos que pasó.
```

Encapsulando código

- Funciones y más funciones
- Clases, o como tratar de modelar la realidad
- Módulos y paquetes

Funciones

Estructura básica

```
>>> def alcuadrado(n):
...         res = n ** 2
...         return res
...
>>> alcuadrado(3)
9
```

Las funciones son objetos

```
>>> alcuadrado
<function alcuadrado at 0xb7c30b54>
>>> f = alcuadrado
>>> f(5)
25
```

Más Funciones

Tengo mucha flexibilidad con los argumentos

```
>>> def func(a, b=0, c=7):
...         return a, b, c
...
>>> func(1)
(1, 0, 7)
>>> func(1, 3)
(1, 3, 7)
>>> func(1, 3, 9)
(1, 3, 9)
>>> func(1, c=9)
(1, 0, 9)
>>> func(b=2, a=-3)
(-3, 2, 7)
```

Clases

Armando una clase

```
>>> class MiClase:
...     x = 3
...     def f(self):
...         return 'Hola mundo'
>>> c = MiClase()
>>> c.x
3
>>> c.f()
'Hola mundo
```

Heredando

```
>>> class MiClase(ClasePadre):
>>> class MiClase(ClasePadre, ClaseTio):
```

Mas Clases

```
>>> class Posicion:
...     def __init__(self, x, y):
...         self.x = x
...         self.y = y
...     def distancia(self):
...         dist = math.sqrt(self.x**2 + self.y**2)
...         return dist
>>> p1 = Posicion(3, 4)
>>> p1.x
3
>>> p1.distancia()
5.0
>>> p2 = Posicion(7, 9)
>>> p2.y, p1.y
(9, 4)
```

El Módulo mas paquete

Módulos

- Funciones, o clases, o lo que sea en un archivo.
- Es un .py normal, sólo que lo importamos y usamos.
- Fácil, rápido, funciona.

Tengo un pos.py, con la clase de la filmina anterior:

```
>>> import pos
>>> p = pos.Posicion(2, 3)
>>> p.x
2
```

Paquetes

- Cuando tenemos muchos módulos juntos.
- Usamos directorios, e incluso subdirectorios.

Tres cositas más

- List comprehensions.
- Generadores.
- Espacio de nombres.

Entendiendo de Listas

List comprehensions

```
>>> vec = [3, 7, 12, 0, 3, -13, 45]
>>> [x**2 for x in vec]
[9, 49, 144, 0, 9, 169, 2025]
>>> [x**2 for x in vec if x <= 7]
[9, 49, 0, 9, 169]
```

Son extremadamente útiles

```
>>> sum([x**2 for x in range(1000)])
332833500
>>> len([x for x in range(1000) if (x**2)%2 == 0])
500
```

Generadores

Ejem: Función que nos devuelve una cantidad de algos

```
>>> def fibonacci(limite):
...     valores = []
...     a, b, c = 0, 1, 0
...     while c < limite:
...         valores.append(b)
...         a, b, c = b, a+b, c+1
...     return valores
>>> fibonacci(8)
[1, 1, 2, 3, 5, 8, 13, 21]
>>> t = 0
>>> for i in fibonacci(8): t += i
>>> t
54
```

Seguimos generando

Somos vagos, vamos devolviendo valor por valor

```
Somos vagos, vamos devolviendo valor por valor
>>> def fibonacci_gen(limite):
...         a, b, c = 0, 1, 0
...         while c < limite:
...                 yield b
...                 a, b, c = b, a+b, c+1
>>> fibonacci_gen(8)
<generator object at 0xb7c294ac>
>>> t = 0
>>> for i in fibonacci_gen(99999999999999999):
...         t += i
```

Una gran idea

Hay varios espacios de nombres

- Básicos: local y global.
- Los tienen las funciones, clases, módulos.

¡El mismo ejemplo que antes!

```
>>> import pos
>>> p = pos.Posicion(2, 3)
>>> p.x
2
```

Más útiles de lo que parecen

- Simplifican la estructura a mentalizar.
- Prolijidad, legibilidad, traceabilidad.

Preguntas?

- Todo lo que acabo de decir no puede ser usado en mi contra.
- Basado en la presentación de Introducción a Python por Facundo Batista
- Visiten la página de GUSLaR `http://guslar.usla.org.ar`
- Anótense a la lista de GUSLaR.
- Anótense a la lista de PyAr.
- Nada más...