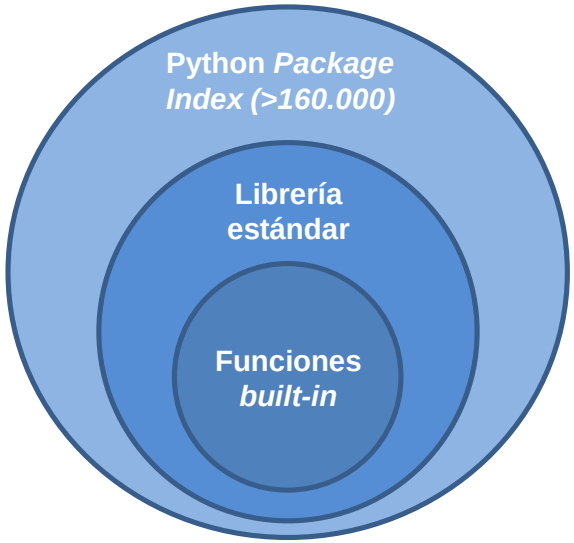
Introducción al lenguaje Python

Los principios

- Algunas de las recomendaciones de PEP-20 (*Python Enhancement Proposals*):
  - *Beautiful is better than ugly.*
  - *Explicit is better than implicit.*
  - *Simple is better than complex.*
  - *Complex is better than complicated.*
  - *Flat is better than nested.*
  - *Sparse is better than dense.*
  - *Readability counts.*
  - …

Las principales características

- Multiparadigma
- Tipado dinámico
- Uso de indentación
- Funciones *built-in*
- Extensa librería estándar (*batteries included*)
- Amplia comunidad de desarrolladores de “librerías de terceros”



Multiparadigma

- Imperativo
- Orientado a objetos
- Funcional

Tipado dinámico

- No se declaran los tipos de los atributos, parámetros o métodos
- *Duck typing*: “Cuando hay algo que camina como un pato, nada como un pato y suena como un pato, a eso se le llama pato”

```
class Pato:
    def parpar(self):
        print "Cuac!"
    def plumas(self):
        print "El pato tiene plumas blancas y grises."

class Persona:
    def parpar(self):
        print "La persona imita el sonido de un pato."
    def plumas(self):
        print "La persona toma una pluma del suelo y la muestra."

def en_el_bosque(pato):
    pato.parpar()
    pato.plumas()

def juego():
    donald = Pato()
    juan = Persona()
    en_el_bosque(donald)
    en_el_bosque(juan)

juego()
```

Instrucciones

```
if a==0:
    b = 1
else:
    b = 2
```

Bloques e indentación

- Las más importantes son:

  - The `if` statement, which conditionally executes a block of code, along with `else` and `elif` (a contraction of else-if).
  - The `for` statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
  - The `while` statement, which executes a block of code as long as its condition is true.
  - The `try` statement, which allows exceptions raised in its attached code block to be caught and handled by `except` clauses; it also ensures that clean-up code in a `finally` block will always be run regardless of how the block exits.
  - The `class` statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.
  - The `def` statement, which defines a function or method.
  - The `with` statement (from Python 2.5), which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing RAII-like behavior.
  - The `pass` statement, which serves as a NOP. It is syntactically needed to create an empty code block.
  - The `assert` statement, used during debugging to check for conditions that ought to apply.
  - The `yield` statement, which returns a value from a generator function. From Python 2.5, `yield` is also an operator. This form is used to implement coroutines.
  - The `import` statement, which is used to import modules whose functions or variables can be used in the current program.
  - The `print` statement was changed to the `print()` function in Python 3.[51]

Comentarios

- De línea:
  - Iniciados con #
- De bloque:
  - Delimitados con ’’’ ó con ”””

Definiciones por compresión

- Permiten definir colecciones de forma muy compacta, al estilo de ciertas descripciones matemáticas de conjuntos:

$$S = \{ \underbrace{2 \cdot x}_{\text{output expression}} \mid \underbrace{x}_{\text{variable}} \in \underbrace{\mathbb{N}}_{\text{input set}} , \underbrace{x^2 > 3}_{\text{predicate}} \}$$

- Se pueden construir de esta forma:
  - Listas
  - Diccionarios
  - Conjuntos

```
def similitud_coseno(v1, v2):
    """Cálculo de la similitud del coseno entre dos vectores"""
    indice = range(0, len(v1))
    v1_v1 = [v1[i]*v1[i] for i in indice]
    v2_v2 = [v2[i]*v2[i] for i in indice]
    v1_v2 = [v1[i]*v2[i] for i in indice]
    return sum(v1_v2)/(sqrt(sum(v1_v1))*sqrt(sum(v2_v2)))
```

Librería estándar

- Incluye cuatro tipos de elementos:
  - *Built-in functions*
  - *Built-in data types*
  - *Built-in exceptions*
  - *Modules*

El grueso de la librería. Accesible mediante import

Funciones predefinidas

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Tipos agregados

- Tuplas (inmutables y valores heterogéneos):
  - (1,2,3,’a’,1)
- Listas (mutables y valores heterogéneos):
  - [1,2,3,’a’,1]
- Conjuntos (mutables y sin repetición):
  - {1,2,4,3,’a’}
- Diccionarios (mutables):
  - {‘rojo’:1, ‘verde’:2, 4:’a’}

Tipos predefinidos

Type	Mutable	Description	Syntax example

`str`	Immutable	A character string: Sequence of Unicode codepoints.	`'Wikipedia'`
			`"Wikipedia"`
			`"""Spanning multiple lines"""`

`bytearray`	Mutable	Sequence of bytes.	`bytearray(b'Some ASCII')`
			`bytearray(b"Some ASCII")`
			`bytearray([119, 105, 107, 105])`

`bytes`	Immutable	Sequence of bytes.	`b'Some ASCII'`
			`b"Some ASCII"`
			`bytes([119, 105, 107, 105])`

`list`	Mutable	List, can contain mixed types.	`[4.0, 'string', True]`
`tuple`	Immutable	Can contain mixed types.	`(4.0, 'string', True)`
`set`	Mutable	Unordered set, contains no duplicates. Can contain mixed types as long as they are hashable.	`{4.0, 'string', True}`
`frozenset`	Immutable	Unordered set, contains no duplicates. Can contain mixed types as long as they are hashable.	`frozenset([4.0, 'string', True])`
`dict`	Mutable	Associative array (or dictionary) of key and value pairs. Can contain mixed types (keys and values). Keys must be a hashable type.	`{'key1': 1.0, 3: False}`
`int`	Immutable	Integer of unlimited magnitude.[59]	`42`
`float`	Immutable	Floating point number (system-defined precision).	`3.1415927`
`complex`	Immutable	Complex number with real and imaginary parts.	`3+2.7j`
`bool`	Immutable	Boolean value.	`True` `False`
`ellipsis`		An ellipsis placeholder to be used as an index in NumPy arrays.	`...`

Librerías de terceros

python™

search

» Package Index

PACKAGE INDEX
Browse packages
List trove classifiers
RSS (latest 40 updates)
RSS (newest 40 packages)
Terms of Service
PyPI Tutorial
PyPI Security
PyPI Support
PyPI Bug Reports
PyPI Discussion
PyPI Developer Info
ABOUT
NEWS
DOCUMENTATION
DOWNLOAD
COMMUNITY
FOUNDATION
CORE DEVELOPMENT

PyPI - the Python Package Index

The Python Package Index is a repository of software for the Python programming language. There are currently **130057** packages here.
To contact the PyPI admins, please use the Support or Bug reports links.

Not Logged In
Login
Register
Lost Login?
Login with OpenID
Login with Google

Status
Spam on PyPI: monitoring

Get Packages

To use a package from this index either "pip install *package*" (get pip) or download, unpack and "python setup.py install" it.

Package Authors

Submit packages with "python setup.py upload". You can also use twine! The index hosts package docs. You must register. Testing? Use testpypi.

Infrastructure

To interoperate with the index use the JSON, XML-RPC or HTTP interfaces. Use local mirroring or caching to make installation more robust.

Updated	Package	Description
2018-02-21	iqos4 0.0.2	Support functions for iqos
2018-02-21	m2tools 0.1.7	Magento 2 python toolkit
2018-02-21	yaramod 1.1.0b1	Library for manipulation of YARA files.
2018-02-21	glmdisc 0.0.1	Supervised multivariate discretization and levels merging for logistic regression
2018-02-21	pyfma 0.1.0	Fused multiply-add for Python
2018-02-21	jsonplus 0.7.0	Custom datatypes (like datetime) serialization to/from JSON.
2018-02-21	sealights-python-agent 0.2.79	Python Agent
2018-02-21	stylelens-crawl-amazon 0.0.13	stylelens-crawl-amazon
2018-02-21	django-ar-organizations 0.2.41	Group accounts for Django
2018-02-21	django-quicky 0.7.1	A collection of tools to make setting up Django quicker.
2018-02-21	mxnet-cu91 1.2.0b20180221	MXNet is an ultra-scalable deep learning framework. This version uses CUDA-9.1.
2018-02-21	waves-core 1.1.8.5	WAVES - core package
2018-02-21	datapackage-pipelines 1.6.7	{{ DESCRIPTION }}
2018-02-21	ldap2pg 4.5	Synchronize PostgreSQL roles and ACLs from LDAP
2018-02-21	clickhouse-mysql 0.0.2018022101	MySQL to ClickHouse data migrator
2018-02-21	django-maintenance-mode 0.7.2	django-maintenance-mode shows a 503 error page when maintenance-mode is on.
2018-02-21	mxnet-cu80 1.2.0b20180221	MXNet is an ultra-scalable deep learning framework. This version uses CUDA-8.0.
2018-02-21	mxnet-cu75 1.2.0b20180221	MXNet is an ultra-scalable deep learning framework. This version uses CUDA-7.5.
2018-02-21	markdown-swiss 0.0.1	This is an extension to Python-Markdown which provides a variety of functions for running python code
2018-02-21	dpsim 0.1.3	DPsim is a real-time power system simulator that operates in the dynamic phasor as well as electromagentic transient domain.
2018-02-21	hepcrawl 9.0.13	Scrapy project for feeds into INSPIRE-HEP (http://inspirehep.net).
2018-02-21	mpf 0.50.0.dev65	Mission Pinball Framework
2018-02-21	tilecloud 0.5.1	Tools for managing tiles
2018-02-21	mxnet-cu90 1.2.0b20180221	MXNet is an ultra-scalable deep learning framework. This version uses CUDA-9.0.
2018-02-21	userdata-cool 0.1.2	Let you write user data easily
2018-02-21	dpp-runner 0.0.6	pipeline runner lib and server
2018-02-21	fincalendar 0.1.8	Finance Calendar
2018-02-21	cudnnenv 0.6.2	cudnn environment manager
2018-02-21	missinglink-kernel 0.3863	Kernel SDK for streaming realtime metrics to https://missinglink.ai
2018-02-21	missinglink-sdk 0.3863	SDK for streaming realtime metrics to https://missinglink.ai
2018-02-21	ccommon 1.1.2	this is a package of common func
2018-02-21	stmpy 0.2.9	Support for simple state machines
2018-02-21	lifxlan 1.1.10	API for local communication with LIFX devices over a LAN.

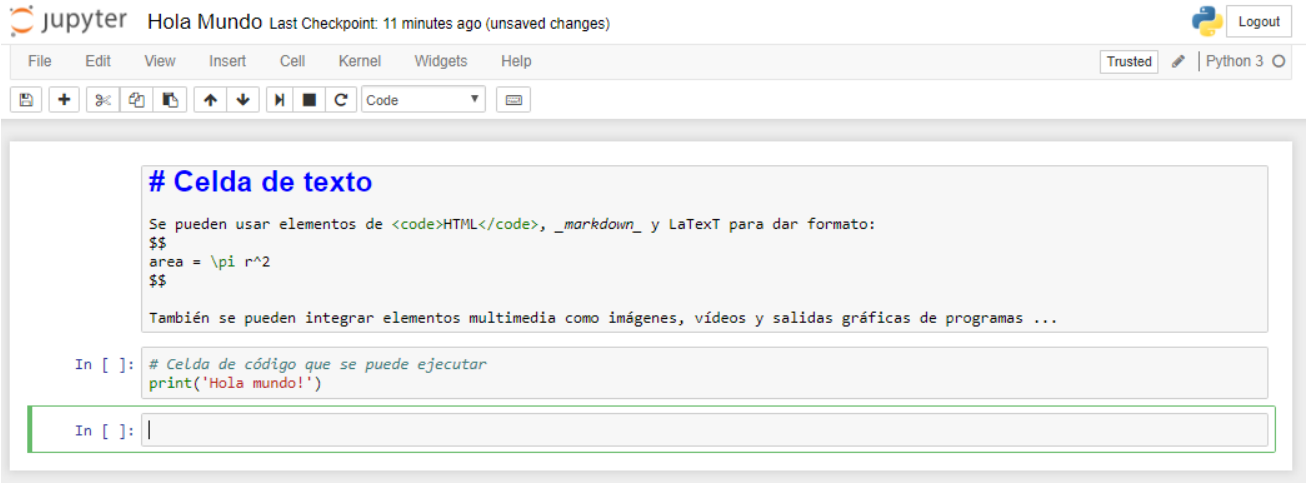
Comprobación en tiempo de ejecución

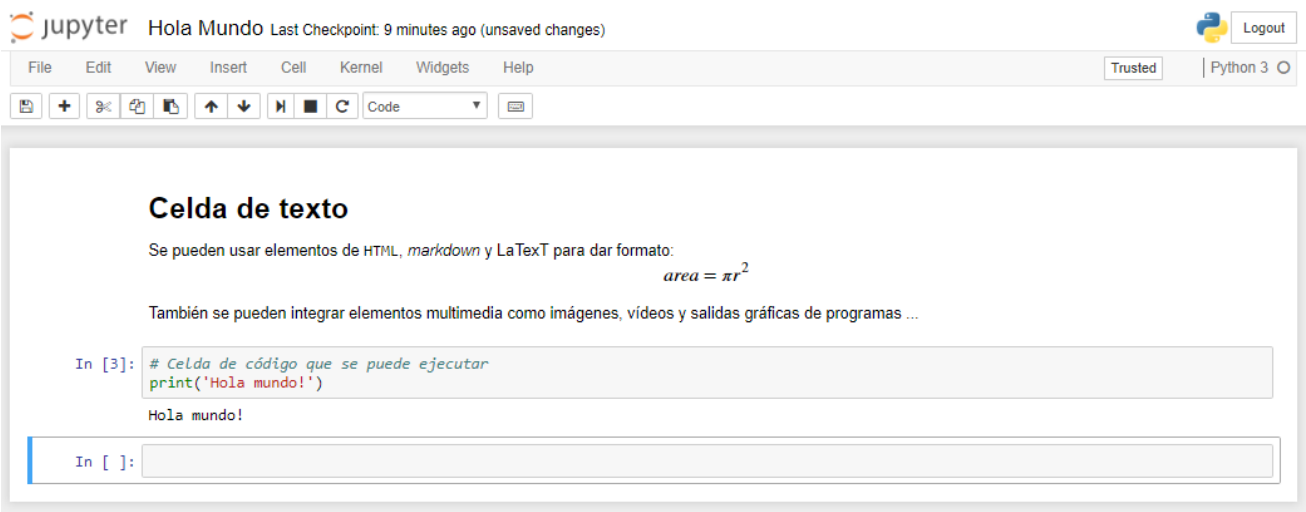
```
def repite(mensaje):
    return (mensaje + mensaje)
```

```
texto = input()
if texto == 'Falla':
    print(repiiiite(texto))
else:
    print(repite(texto))
```

Funciona salvo que la entrada sea `'Falla'`

Nuestro entorno de trabajo: *notebooks* de Jupyter





Tarea

- Trabajar con el notebook:
  - “Recetas simples de Python.ipynb”

Funciones

```
[ ]: # Definición de la función doble, que devuelve el parámetro recibido multiplicado por 2

[ ]: # Llamada a la función doble con el parámetro 2

[ ]: # Llamada a la función doble con el parámetro 'Hola '

[ ]: # Llamada a la función doble con el parámetro [1, 's', 'hola']

[ ]: # Definición de la función doble_lista mediante iteración. La función recibe una lista como parámetro,
     # y devuelve otra lista en el que cada elemento de la lista original ha sido multiplicado por 2.

[ ]: # Llamada a la función doble_lista con el parámetro [1, 4, 3]

[ ]: # Definición de la funcion doble_lista mediante comprensión.

[ ]: # Definición de la funcion aplica, que recibe los siguientes parámetros:
     #   - Una lista
     #   - Una función (por defecto tomará el valor doble)
     # La función aplica devuelve una lista en la que se ha aplicado la funcion a cada elemento de la lista original
```