Introducción a Python

Héctor Enríquez

Outline

Números Enteros

```
a = 1
print(a)
a += 1
print(a)
a = a + 1
print(a)
```

```
1
2
3
```

Conversión a enteros int

```
print(int(1), int("1"), int(1.2))
```

```
1 1 1
```

Cadenas de caracteres

```
a = "texto de ejemplo"
print(a)
a += " final"
print(a)
a = "#" * 10
print(a)
```

```
texto de ejemplo
texto de ejemplo final
##########
```

Conversión a cadenas str

```
print((str("3"), str(3), str(3.2)))
```

```
('3', '3', '3.2')
```

Números Reales

```
a = 6.4
print(a)
a /= 2
print(a)
```

```
6.4
3.2
```

Conversión a reales `float`

```
print(float(3.2), float("3"), float(3))
```

```
3.2 3.0 3.0
```

Listas

```
a = [1, 2, 3]
print(a)
print(a + [8, 3])
print(a * 3)
```

```
[1, 2, 3]
[1, 2, 3, 8, 3]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Convertir a lista `list`

```
print(list("123"), list((1, 2, 3)), list({1: "a", 2: "b", 3: "c"}))
```

```
['1', '2', '3'] [1, 2, 3] [1, 2, 3]
```

Diccionarios

```
a = {"clave1": "valor1", "clave2": "valor2", 3: "otro", "ultimo": 4}
print(a)
```

Listas sin abreviar y abreviando

```
lista = []
for i in range(10):
    lista.append(i**2)
print(lista)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
lista = [i**2 for i in range(10)]
print(lista)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Diccionarios sin abreviar y abreviando

```
squares = {}
for i in range(10):
    squares[i] = i**2
print(squares)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

```
squares = {i: i**2 for i in range(10)}
print(squares)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

Tuplas sin abreviar y abreviando

```
tupla = tuple(i**2 for i in range(10))
print(tupla)
```

```
(0, 1, 4, 9, 16, 25, 36, 49, 64, 81)
```

Generadores

```
generador = (i**2 for i in range(10))
print(generador)
```

```
<generator object <genexpr> at 0x6ffffcab620>
```

```
generador = (i**2 for i in range(10))
for i in generador:
    print(i, end=", ")
for i in generador:
    print(i)
```

```
0, 1, 4, 9, 16, 25, 36, 49, 64, 81,
```

Generadores II

```
from sys import getsizeof
generador = (i**2 for i in range(10))
lista = [i**2 for i in range(10)]
print(getsizeof(generador), getsizeof(lista))
```

```
88 192
```

```
from sys import getsizeof
generador = (i**2 for i in range(100))
lista = [i**2 for i in range(100)]
print(getsizeof(generador), getsizeof(lista))
```

```
88 912
```

Generadores III

```
from timeit import timeit
print(timeit("sum(i**2 for i in range(100))", number=1000))
print(timeit("sum([i**2 for i in range(100)])", number=1000))
```

```
0.030426119999901857
0.029341689001739724
```

def

```
def fun(a, b, c, d=1, e=""):
    print(a, b, c, d, e)
```

```
def fun(*args, **kwargs):
    print(args, kwargs)
    return args, kwargs
```

```
def fun(arg, *args, kwarg="default", **kwargs):
    print(arg, args, kwarg, kwargs)
```

def II

Peligroso

```
def fun(arg=[]):
    arg.append(len(arg))
    print(arg)

fun()
fun()
```

```
[0]
[0, 1]
```

lambda

```
fun = lambda x: x**2
print(fun(10))
```

```
100
```

map

```
lista = [i**2 for i in range(10)]
cadenas = map(str, lista)
print(cadenas)
print(list(cadenas))
```

```
<map object at 0x6ffffcb0d68>
['0', '1', '4', '9', '16', '25', '36', '49', '64', '81']
```

filter

```
lista = [i**2 for i in range(10)]
positivos = filter(lambda x: x > 0, lista)
print(positivos)
print(list(positivos))
```

```
<filter object at 0x6ffffcb0d30>
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

reduce

```
from functools import reduce
lista = [i**2 for i in range(10)]
resta = reduce((lambda x, y: x - y), lista)
print(resta)
```

```
-285
```

Comprobando el tipo del parámetro arg

```
def plus1(arg):
    if isinstance(arg, int):
        return arg + 1
    elif isinstance(arg, str):
        return int(arg) + 1
    elif isinstance(arg, list):
        return list(map(lambda x: x + 1, arg))

print(plus1(2))
print(plus1("2"))
print(plus1([2, 3]))
```

```
3
3
[3, 4]
```

Comprobando el tipo dentro del parámetro

```
def plus1(arg):
    if isinstance(arg, int):
        return arg + 1
    elif isinstance(arg, str):
        return int(arg) + 1
    elif isinstance(arg, list):
        return list(map(plus1, arg))

print(plus1(2))
print(plus1("2"))
print(plus1([2, 3]))
```

```
3
3
[3, 4]
```

Definiendo una clase

```
class Clase:
    def __init__(self):
        self.variable1 = "cadena"
        self.variable2 = 20

    def metodo1(self, a, b):
        self.variable1 = a
        self.variable2 = b
        return self.variable1 + self.variable2
```

Herencia

```cpp
#include <iostream>
class A {
  public:
    A() {std::cout << "constructor A" << std::endl; f();}
    virtual void f() {std::cout << "A::f()" << std::endl;}};
class B : public A {
  public:
    B() {std::cout << "constructor B" << std::endl; f();}
    virtual void f() {std::cout << "B::f()" << std::endl;}};
int main(int argc, char *argv[]){
    A a = A(); B b = B(); return 0;}
```

```
constructor A
A::f()
constructor A
A::f()
constructor B
B::f()
```

Herencia II

```
class A:
    def __init__(self):
        print("constructor A")
        self.f()
    def f(self):
        print("A.f()")
class B(A):
    def __init__(self):
        print("constructor B")
        self.f()
    def f(self):
        print("B.f()")
A(); B()
```

```
constructor A
A.f()
constructor B
B.f()
```

Herencia III

```
class A:
    def __init__(self):
        print("constructor A")
        self.f()
    def f(self):
        print("A.f()")
class B(A):
    def __init__(self):
        super().__init__()
        print("constructor B")
        self.f()
    def f(self):
        print("B.f()")
A(); B()
```

Herencia IV

```
constructor A
A.f()
constructor A
B.f()
constructor B
B.f()
```

- No tiene sentido una clase base con métodos virtuales por ser...
  - Sí una clase base con métodos comunes que no se van a modificar
- Conseguir que funcione como los otros lenguajes cuesta mucho y no es coherente

¿Qué es un módulo y para qué sirve?

- Ficheros y Directorios **empaquetados**
- Los ficheros contienen clases y funciones
- Código que define el comportamiento como **paquete**
  - *__init__.py*
- Engloban servicios y funcionalidades
- Permiten hacer scripts en pocas líneas
  - Es más rápido buscar un módulo que una solución

Importación

```
import os
from os import path
import os.path as osp
```

- Se pueden importar:
  - Variables
  - Funciones
  - Clases
- Todas conservarán su propio espacio de nombres

re

Operaciones con expresiones regulares

```
from re import match, search, findall
texto = "Vamos con afán todos a la vez a buscar con ainco.."
if match("Vamos", texto):
    print("Empieza por 'Vamos'")
if search("afán", texto):
    print("Contiene 'afán'")
print(findall(" ([a-zA-Z]{3}) ", texto))
```

```
Empieza por 'Vamos'
Contiene 'afán'
['con', 'vez', 'con']
```

- Regex no es idéntico en todos los lenguajes pero las diferencias son mínimas
- Si se va a usar varias veces un mismo regex conviene compilarlo `re.compile`
- Una de las librerías/paquetes más ampliamente usados automatizando tareas

re reference

time

Manejo del reloj del sistema y conversiones

```
from time import gmtime, strftime, strptime
t = gmtime()
print(t)
print(strftime("%a, %d %b %Y %H:%M:%S +0000", t))
print(strptime("30 Nov 00", "%d %b %y"))
```

```
time.struct_time(tm_year=2018, tm_mon=10, tm_mday=21, tm_hour=9, tm_min=20, tm_sec=50, tm_wday=6, tm_
Sun, 21 Oct 2018 09:20:50 +0000
time.struct_time(tm_year=2000, tm_mon=11, tm_mday=30, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3, tm_yd
```

time reference

datetime

Tipos de fecha y hora básicos

```
from datetime import datetime
print(datetime.now())
print(datetime.strptime("2018-09-08  10:15:54.123456",
                        '%Y-%m-%d  %H:%M:%S.%f'))
print(repr(datetime.strptime("2018-09-08  10:15:54.123456",
                             '%Y-%m-%d  %H:%M:%S.%f')))
```

```
2018-10-20 19:23:00.768618
2018-09-08 10:15:54.123456
datetime.datetime(2018, 9, 8, 10, 15, 54, 123456)
```

datetime

math

```
from math import fsum, exp, log, tan, atan
f = [.1, .1, .1, .1, .1, .1, .1, .1, .1, .1]
print(sum(f))
print(fsum(f))
print(log(exp(2.53)))
print(tan(atan(1.71)))
```

```
0.9999999999999999
1.0
2.53
1.7100000000000002
```

math reference

itertools

Creación, manipulación y combinación de iteradores

```
from itertools import chain, product
print(list(chain("ABC", "DEFG")))
print(list(product("123", "ABC")))
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G']
[('1', 'A'), ('1', 'B'), ('1', 'C'), ('2', 'A'), ('2', 'B'), ('2', 'C'), ('3', 'A'), ('3', 'B'), ('3
```

itertools reference

functools

Funciones que operan sobre funciones - Decoradores

```
from functools import lru_cache, partial
@lru_cache(maxsize=1024)
def heavy(integer):
    integers = []
    for i in range(1, integer):
        if integer % i == 0:
            integers.append(i)
    return integers
from timeit import Timer
t = Timer(partial(heavy, 123456))
print(t.timeit(1))
print(t.timeit(1))
```

```
0.007704327999817906
9.64000264502829e-07
```

functools II

```
from functools import lru_cache
@lru_cache(maxsize=1024)
def bug():
    from datetime import datetime
    return datetime.now()
print(bug())
print(bug())
```

```
2018-10-21 21:57:13.871278
2018-10-21 21:57:13.871278
```

os

Batiburrillo de interfaces con el sistema operativo

```
import os
from os.path import join, getsize
for root, dirs, files in os.walk('python/Lib/email'):
    print(root, "consumes", end=" ")
    print(sum(getsize(join(root, name)) for name in files), end=" ")
    print("bytes in", len(files), "non-directory files")
    if 'CVS' in dirs:
        dirs.remove('CVS')  # don't visit CVS directories
```

os reference

subprocess

Lanzamiento, control y explotación de procesos hijos.

```
from subprocess import Popen, PIPE
process = Popen(['echo', 'parametros'], stdout=PIPE, stderr=PIPE)
stdout, stderr = process.communicate()
print(stdout)
```

```
b'parametros\n'
```

subprocess reference

Comparación SQL y NoSQL

SQL cumplen ACID

- Atomicidad: si una de un grupo falla recupero antes de la transacción.
- Consistencia: todos los nodos devuelven lo mismo o fallan.
- Independencia: podemos aplicar varias transacciones al mismo conjunto de datos.
- Durabilidad: una vez hecho el commit siempre puedo recuperar la información.

NoSQL cumplen dos CAP

- Consistencia (Consistency): que todos los nodos vean la misma información al mismo tiempo.
- Disponibilidad (Availability): la garantía de que cada petición a un nodo reciba una confirmación de si ha sido o no resuelta satisfactoriamente.
- Tolerancia al particionado (Partition Tolerance): el sistema sigue funcionado incluso si algunos nodos fallan.

SQLite

```
import sqlite3
connection = sqlite3.connect("company.db")
cursor = connection.cursor()
sql_command = """
CREATE TABLE employee (staff_number INTEGER PRIMARY KEY, fname VARCHAR(20),
lname VARCHAR(30), gender CHAR(1), joining DATE, birth_date DATE);"""
cursor.execute(sql_command)

sql_command = """INSERT INTO employee (staff_number, fname, lname, gender,
  birth_date) VALUES (NULL, "William", "Shakespeare", "m", "1961-10-25");"""
cursor.execute(sql_command)
sql_command = """INSERT INTO employee (staff_number, fname, lname, gender,
  birth_date) VALUES (NULL, "Frank", "Schiller", "m", "1955-08-17");"""
cursor.execute(sql_command)

connection.commit()
connection.close()
```

SQLite II

```python
import sqlite3
connection = sqlite3.connect("company.db")
cursor = connection.cursor()
cursor.execute("SELECT * FROM employee")
print("  fetchall:")
result = cursor.fetchall()
for r in result:
    print(r)
cursor.execute("SELECT * FROM employee")
print("  fetch one:")
res = cursor.fetchone()
cursor.execute("""DROP TABLE employee;""")
print(res)
```

```
  fetchall:
(1, 'William', 'Shakespeare', 'm', None, '1961-10-25')
(2, 'Frank', 'Schiller', 'm', None, '1955-08-17')
  fetch one:
(1, 'William', 'Shakespeare', 'm', None, '1961-10-25')
```

MongoDB

```
pip3 install mongoengine
```

```
from mongoengine import *
connect('mydb')

class BlogPost(Document):
    title = StringField(required=True, max_length=200)
    posted = DateTimeField(default=datetime.datetime.utcnow)
    tags = ListField(StringField(max_length=50))
    meta = {'allow_inheritance': True}

class TextPost(BlogPost):
    content = StringField(required=True)

class LinkPost(BlogPost):
    url = StringField(required=True)
```

MongoDB II

```
# Create a text-based post
>>> post1 = TextPost(title='Using MongoEngine', content='See the tutorial')
>>> post1.tags = ['mongodb', 'mongoengine']
>>> post1.save()

# Create a link-based post
>>> post2 = LinkPost(title='MongoEngine Docs', url='hmarr.com/mongoengine')
>>> post2.tags = ['mongoengine', 'documentation']
>>> post2.save()

# Iterate over all posts using the BlogPost superclass
>>> for post in BlogPost.objects:
...     print('===', post.title, '===')
...     if isinstance(post, TextPost):
...         print post.content
...     elif isinstance(post, LinkPost):
...         print 'Link:', post.url
...     print
...
```

MongoDB III

```
# Count all blog posts and its subtypes
>>> BlogPost.objects.count()
2
>>> TextPost.objects.count()
1
>>> LinkPost.objects.count()
1

# Count tagged posts
>>> BlogPost.objects(tags='mongoengine').count()
2
>>> BlogPost.objects(tags='mongodb').count()
1
```

Bucle de eventos

Futuros

Hilos

Procesos

async y await

```python
import asyncio

async def fetch():
    await asyncio.sleep(10)
    return "hola"

async def main():
        result = await fetch()
        print(result)

if __name__ == '__main__':
    asyncio.run(main())
```

```
hola
```

async y await II

```python
import asyncio
from datetime import datetime
from time import sleep

async def fetch(delay):
    await asyncio.sleep(delay)
    # sleep(delay)
    return "sleep " + str(delay) + ", " + str(datetime.now())

async def main():
    delays = [i/10 for i in range(1, 10)]
    results = await asyncio.gather(*[
        asyncio.create_task(fetch(delay))
        for delay in delays])
    for result in results:
        print(result)

if __name__ == '__main__':
    asyncio.run(main())
```

async y await III

```
sleep 0.1, 2018-10-27 11:11:29.536400

sleep 0.2, 2018-10-27 11:11:29.626400

sleep 0.3, 2018-10-27 11:11:29.739400

sleep 0.4, 2018-10-27 11:11:29.827400

sleep 0.5, 2018-10-27 11:11:29.938400

sleep 0.6, 2018-10-27 11:11:30.039400

sleep 0.7, 2018-10-27 11:11:30.131400

sleep 0.8, 2018-10-27 11:11:30.230400

sleep 0.9, 2018-10-27 11:11:30.334400
```

async y await IV

Instalando paquetes

- Linux

```
pip3 install aiohttp
```

- Windows

```
pip install aiohttp
```

Comienzo del proyecto

```python
import aiohttp; import asyncio

async def fetch(session, url):
    async with session.get(url) as response:
        if response.status != 200:
            response.raise_for_status()
        return await response.text()

async def fetch_all(session, urls):
    results = await asyncio.gather(*[asyncio.create_task(fetch(session, url)) for url in urls])
    return results

async def main():
    urls = ['http://cnn.com', 'http://google.com']
    async with aiohttp.ClientSession() as session:
        htmls = await fetch_all(session, urls)
        for html in htmls: print(html.encode("utf-8"))

if __name__ == '__main__': asyncio.run(main())
```

Mutex

Multiprocessing

Pautas de programación

- Evitar el estado compartido
- Picklability
- Thread safety of proxies
- Unir explícitamente todos los procesos (`join`)
- Mejor heredar que pickle / unpickle
- Evitar `terminate` con procesos que comparten objetos
- Unir procesos que comparten colas
- Pasar explícitamente todos los recursos
- Reemplazar `sys.stdin` por un *file descriptor*
- Importación segura con `if __name__ == '__main__':`

Process

```python
from multiprocessing import Process

def f(name):
    print('hello', name)

if __name__ == '__main__':
    p = Process(target=f, args=('bob',))
    print("pre-start")
    p.start()
    print("post-start")
    p.join()
```

```
pre-start
post-start
hello bob
```

Poll

```python
from datetime import datetime
from multiprocessing import Pool

def f(number):
    return "number " + str(number) + ", " + str(datetime.now())

if __name__ == '__main__':
    with Pool(5) as p:
        results = p.map(f, range(10))
    for result in results:
        print(result)
```

Poll II

```
number 0, 2018-10-25 17:22:50.504911
number 1, 2018-10-25 17:22:50.505911
number 2, 2018-10-25 17:22:50.505911
number 3, 2018-10-25 17:22:50.505911
number 4, 2018-10-25 17:22:50.505911
number 5, 2018-10-25 17:22:50.505911
number 6, 2018-10-25 17:22:50.505911
number 7, 2018-10-25 17:22:50.505911
number 8, 2018-10-25 17:22:50.505911
number 9, 2018-10-25 17:22:50.505911
```

concurrent.futures

```python
from datetime import datetime
from concurrent.futures import ProcessPoolExecutor

def f(number):
    return "number " + str(number) + ", " + str(datetime.now())

if __name__ == '__main__':
    with ProcessPoolExecutor(max_workers=5) as ex:
        results = ex.map(f, range(10))
    for result in results:
        print(result)
```

concurrent.futures II

```
number 0, 2018-10-25 17:34:37.328511
number 1, 2018-10-25 17:34:37.328511
number 2, 2018-10-25 17:34:37.328511
number 3, 2018-10-25 17:34:37.344111
number 4, 2018-10-25 17:34:37.344111
number 5, 2018-10-25 17:34:37.344111
number 6, 2018-10-25 17:34:37.344111
number 7, 2018-10-25 17:34:37.344111
number 8, 2018-10-25 17:34:37.344111
number 9, 2018-10-25 17:34:37.344111
```

concurrent.futures III

```
from datetime import datetime
from concurrent.futures import ProcessPoolExecutor, as_completed

def f(number):
    sleep(10 - number)
    return "number " + str(number) + ", " + str(datetime.now())

if __name__ == '__main__':
    with ProcessPoolExecutor(max_workers=5) as ex:
        results = [ex.submit(f, i) for i in range(10)]
        for result in as_completed(results):
            print(result.result())
```

concurrent.futures IV

```
number 4, 2018-10-27 11:17:15.333400
number 3, 2018-10-27 11:17:16.242400
number 2, 2018-10-27 11:17:17.209400
number 1, 2018-10-27 11:17:18.201400
number 0, 2018-10-27 11:17:19.131400
number 9, 2018-10-27 11:17:20.131400
number 8, 2018-10-27 11:17:20.201400
number 7, 2018-10-27 11:17:20.209400
number 6, 2018-10-27 11:17:20.242400
number 5, 2018-10-27 11:17:20.333400
```