programa de BECAS
DGTIC
de FORMACIÓN en TIC

Seguridad Informática

Programación con Python

Servando Miguel López Fernández
Jesús Enrique Pacheco Franco

Modulo argparse

Es un modulo que nos permite interfaces de línea de comandos de manera sencilla.

Dentro de nuestro programa especificamos que argumentos requerimos y argparse se encargara de parsear sys.argv para recolectarlos.

argparse genera mensajes de ayuda y uso de manera automática y también proporciona mensajes de error cuando se proporcionan argumentos no esperados.

Creando el parser

El primer paso es crear un objeto del tipo ArgumentParser.

Este objeto almacenara toda la información necesaria para parsear la información de la línea de comandos en tipos de dato de python3.

```
parser = argparse.ArgumentParser(
    prog='PROG',
    description='''this description
        was indented weird
            but that is okay''',
    epilog='''
            likewise for this epilog whose whitespace will
        be cleaned up and whose words will be wrapped
        across a couple lines''')
parser.print_help()
```

Descargar argparse.py

Añadiendo y parseando argumentos

Para añadir argumentos a nuestro objeto **ArgumentParser** se hace uso del método **add_argument()**.

```
ArgumentParser.add_argument(name or flags...[, action][, nargs][, const][, default][, type][, choices][, required][, help][, metavar][, dest])
```

Para parsear los argumentos simplemente se tiene que llamar al método **parse_args()**.

```
parser.parse_args()
```

name or flags - Either a name or a list of option strings, e.g. `foo` or `-f, --foo`.
action - The basic type of action to be taken when this argument is encountered at the command line.
nargs - The number of command-line arguments that should be consumed.
const - A constant value required by some action and nargs selections.
default - The value produced if the argument is absent from the command line.
type - The type to which the command-line argument should be converted.
choices - A container of the allowable values for the argument.
required - Whether or not the command-line option may be omitted (optionals only).
help - A brief description of what the argument does.
metavar - A name for the argument in usage messages.
dest - The name of the attribute to be added to the object returned by `parse_args()`.

- Documentación official de argparse

https://docs.python.org/3/library/argparse.html#required

Ejercicio #05

Crear un script que reciba un archivo y elimine todos los espacios en blanco al principio y al final de cada línea, además el script deberá lograr que las palabras de cada línea estén separadas únicamente por un espacio. Una vez limpias las líneas deberán ser escritas en otro archivo.

El script deberá recibir el nombre del archivo a limpiar **'–in-file'** y el archivo destino **'–out-file'** utilizando argparse además si se indica la bandera **'–upper'** todo el contenido deberá ser escrito en mayúsculas y si se indica la bandera **'–lower'** todo el contenido deberá ser escrito en minúsculas.

Descargar: texto_sucio.txt

Programación funcional

- Paradigma de programación que (como bien dice su nombre), radica en el uso de funciones.
- Las **funciones de orden superior** son la parte central de este paradigma.
- Las funciones de orden superior reciben otras funciones como argumentos, o regresan otras funciones como resultados.

Programación funcional

- Pasar una función como argumento:

```
def aplica_dos_veces(func,arg):
        return func(func(arg))

def suma_cinco(x):
        return x + 5

print (aplica_dos_veces(suma_cinco,10) ) ->  20
```

Programación funcional

- ¿Qué imprime el siguiente código?:

```
def test(func, arg):
        return func(func(arg))

def mult(x):
        return x * x

print test(mult, 2) -> ___
```

Funciones puras e impuras

- La programación funcional busca utilizar funciones “puras”.
- Una función pura no tiene efectos secundarios y devuelven un valor que dependen **únicamente** de sus argumentos.
- Se habla de pureza pues son como las funciones matemáticas, cos(x) siempre devolverá el mismo resultado para el mismo valor de x.

Funciones puras e impuras

```
lista_1 = [‘a1’,’a2’,’a3’]

def func_pura(x,y):
    temp = x + 2*y
    return temp / (2*x+y)

def func_impura(arg):
    lista_1.append(arg)
```

Funciones puras e impuras

¿Son puras estas funciones?

```
def func(x):
    y = x ** 2
    z = x + y
    z *= 2
    return x
```

```
def func(x, lista1):
    lista1.sort()
    lista1.append(x)
    return lista
```

Funciones puras e impuras

- Utilizar funciones puras, como todo, tiene sus ventajas y desventajas.
- “Más fáciles de analizar y probar”
- Más fáciles de ejecutar en paralelo
- La principal desventaja de utilizar funciones puras es que complican en gran medida operaciones de I/O, pues inherentemente tienen efectos secundarios.
- En algunos casos pueden ser más difíciles de escribir.

Funciones Lambda

- Crear una función (def) asigna una variable automáticamente (el nombre de la función).
- Objetos como las cadenas y números pueden ser creados sin ser asignados a una variable.
- Una función puede ser “anónima” (sin ser asignada a una variable), si se usa la notación lambda.
- Las funciones lambda no son tan potentes como las funciones nombradas, generalmente sirven para soluciones de operaciones simples.
- No pueden contener ciclos (RECURSIVIDAD!)

Funciones Lambda

- Función con nombre

```
def func1(x):
        return x**2 + 5*x + 4
print func1(10)
```

- Función anónima

```
print ((lambda x: x**2 + 5*x + 4)(10))
```

Funciones Lambda

- Función lambda que eleva al cuadrado el argumento

```
(lambda x: x**2) (10)
```

- Función lambda que multiplica dos números

```
(lambda x,y: x*y) (4,5)
```

- Función lambda que valida un palíndromo

```
(lambda x: x == x[::-1]) (‘anitalavalatina’)
```

Funciones Lambda

- Es posible asignar una función lambda a una variable. (Aunque esto claramente le quita lo anónimo)

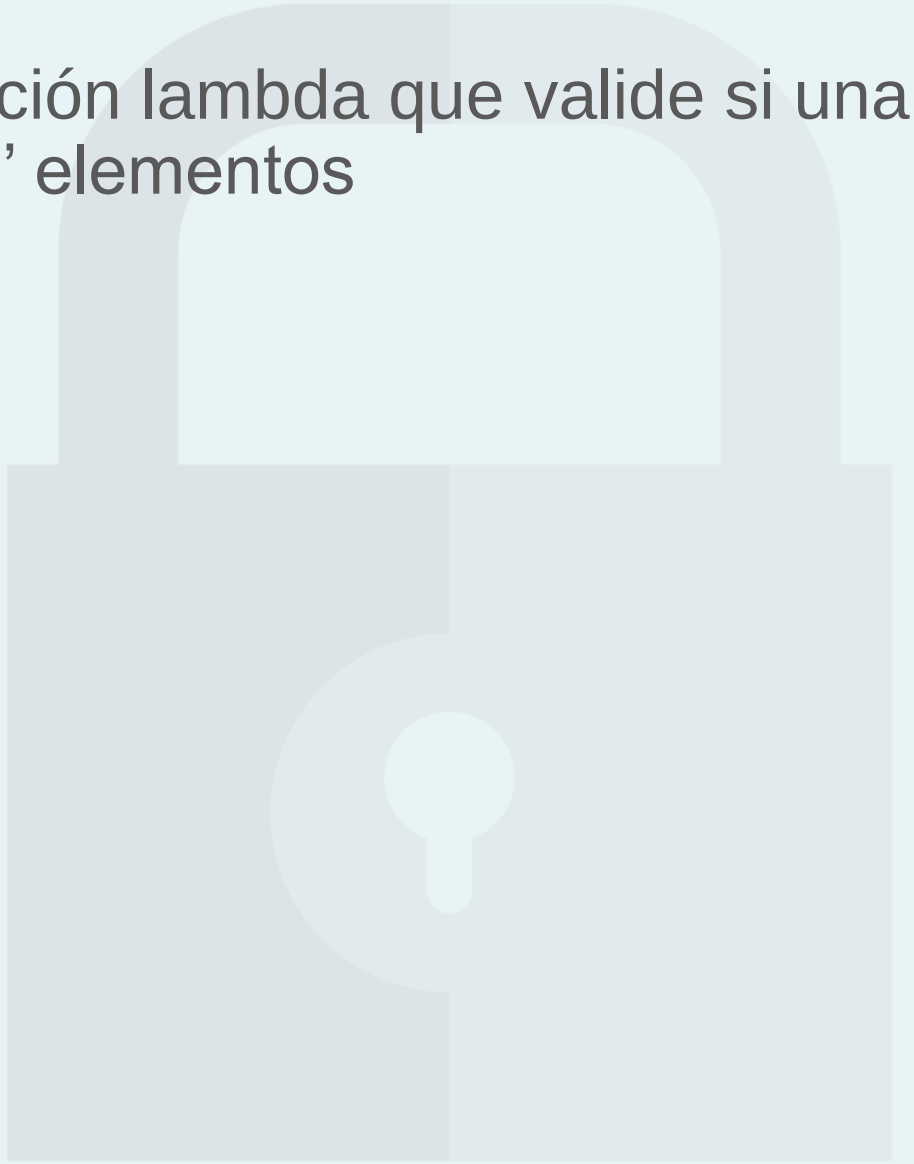
```
pal = (lambda x: x == x[::-1])
pal('anitalavalatina')
```

Funciones Lambda Ejercicios Participación

- Escribir función lambda que multiplique tres números

- Escribir función lambda que valide si una lista está vacía

- Escribir función lambda que valide si una lista tiene al menos ‘n’ elementos



- Escribir función lambda que calcule la raíz cuadrada de un número

Escribir función lambda que obtenga la intersección de dos conjuntos

Map , Filter y Reduce

- Son funciones de orden superior, pues reciben otra función como primer argumento.
- Como segundo argumento reciben un conjunto de elementos (lista, tupla, conjunto)
- Map aplica la función a todos los elementos del conjunto
- Filter genera una nueva lista con todos los elementos que cumplan una condición
- Reduce aplica una operación a cada par de elementos, guardando el resultado anterior como primer operador de la siguiente operación.
- Es común que la función que se pasa como argumento, sea una función anónima

Map

- Aplica la función a todos los elementos del conjunto

```
op_interna = [‘quintero’,’Fernando’,’yEudiEL’]
map(lambda nombre: nombre.upper(), op_interna)

[‘QUINTERO’,’FERNANDO’,’YEUDIEL’]
```

Filter

- Genera una nueva lista con todos los elementos que cumplan una condición

jefes_area = [‘quintero’,’angie’,’demian’,’juan’,’celica’]

filter(lambda nombre: ‘i’ in nombre, jefes_area)

[‘quintero’,’angie’,’demian’,’celica’]

Reduce

- No es parte de las funciones estándar de python3 por lo cual debemos de importarla.

```
from functools import reduce
```

- Aplica una operación a cada par de elementos, guardando el resultado del anterior como el primer operador de la siguiente operación.

```
num = range(10)
reduce(lambda x,y: x+y,num)
45
```

Ejercicio de clase 6

Expresión Funcional:

1) función lambda que sume las tres listas
2) filtre la lista resultante para obtener a los que tienen un solo nombre (filter)
3) convierta a mayúsculas los nombres del resultado anterior (map)
4) obtener una cadena con los nombres resultantes, separando los nombres con coma (reduce)

UNA SOLA LINEA

Descargar: ejercicio6.py

Biblioteca estándar de Python

- Es un conjunto de módulos que vienen por defecto en cualquier instalación de Python.
- Contienen funciones y clases útiles para una gran cantidad de aplicaciones.
- Algunos módulos de la biblioteca estándar cambian entre Python 2.x y Python 3.x
- Para utilizar los módulos, basta con importarlos al inicio de un script
  - from **modulo** import **funcion1,funcion2**
  - from **modulo** import *
  - import **modulo**
  - import **modulo** as **alias**

Built-in

- Funciones
  - print()
  - open()
  - range()
- Constantes
  - True
  - False
  - None
- Tipos
  - int, float, complex
- Exepciones
  - IndexError, KeyError, NameError

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	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()	

Algunos módulos de la biblioteca estándar

Módulo	Descripción
re	Contiene operaciones comunes para expresiones regulares.
string	Contiene operaciones comunes para cadenas, así como algunos ya definidos (mayúsculas, minúsculas, etc.).
socket	Contiene la clase Socket que implementa métodos para establecer conexiones entre clientes y servidores.
base64	Contiene funciones que permiten codificar y decodificar archivos y cadenas a base64.
datetime	Contiene diversos tipos para fechas y horas (hora actual, fecha actual) y métodos para operar con ellos.
sys	Parámetros y funciones específicas del sistema. Sirve para obtener argumentos de la línea de comandos.
subprocess	Administración de subprocesos. Permite ejecutar comandos propios del sistema operativo.

https://docs.python.org/3/library/index.html

Tarea 4 – parte 1

- Reportar:
  - En salida estándar y en un archivo
  - Hora de ejecución
  - MD5 de archivo xml
  - SHA1 de archivo xml
  - Cantidad de hosts prendidos
  - Cantidad de hosts apagados
  - Cantidad de hosts con puerto 22 abierto
  - Cantidad de hosts con puerto 53 abierto
  - Cantidad de hosts con puerto 80 abierto
  - Cantidad de hosts con puerto 443 abierto
  - Cantidad de hosts que tienen nombre de dominio
  - Servidores HTTP usados
    - Cuántos usan Apache
    - Cuántos honeypots (Dionaea)
    - Cuántos usan Nginx
    - Cuántos usan otros servicios

Tarea 4 – parte 2

- Archivos csv con direcciones IP
  - Hosts apagados
  - Hosts prendidos
  - Hosts con puerto 22 abierto
  - Hosts que son honeypots
  - Hosts que tienen nombre de dominio

  Descargar: nmap.xml