PythonBasicCourse-es007

October 4, 2022

1 Curso básico de Python

1.1 Apuntes

Curso básico de Python. Apuntes por Marcelo Horacio Fortino. Versión 2.2. Agosto 2022.

Versión	Autor/es	Fecha	Observaciones
1.0	Marcelo Horacio Fortino	2021/Marzo	Curso Python
1.1	Marcelo Horacio Fortino	2021/Junio	Convertido a markdown - ipynb
1.2	Marcelo Horacio Fortino	2021/Agosto	Actualizados contenidos
1.3	Marcelo Horacio Fortino	2021/Octubre	Agregado Flask microframework
1.4	Marcelo Horacio Fortino	2021/Noviembre	Agregado Pandas - datascience
1.5	Marcelo Horacio Fortino	2021/Diciembre	Agregado Devops - Ansible
2.0	Marcelo Horacio Fortino	2022/Abril	Nueva estructura: core / module
2.1	Marcelo Horacio Fortino	2022/Junio	Módulo apuntes intermedio
2.2	Marcelo Horacio Fortino	2022/Agosto	Actualizado temario y ejercicios

Estos apuntes se basan en: - La documentación oficial de Python, https://docs.python.org/es/3/tutorial/index.html, - La bibliografía presentada al final de este documento, y - Documentación propia recogida a lo largo de los años de diversas fuentes.

1.2 Objetivos del curso

- Objetivo general del curso:

- Aprender a usar Python para crear scripts y programas simples plenamente funcionales, desarrollar aplicaciones web y realizar análisis de datos.

- Objetivos específicos:

- Reconocer las características principales de Python y su utilidad práctica.
- Identificar tipos de datos (simples y compuestos) y operadores.
- Aplicar variables y estructuras de control de flujo.
- Construir funciones y clases (POO).
- Clasificar los módulos y paquetes por sus funcionalidades y objetivos.
- Comparar el lenguaje con otros similares de scripts, procedimentales y orientados a objetos.
- Probar bibliotecas para conexiones REST a aplicaciones web y bases de datos.
- Resolver problemas y errores en el código fuente proponiendo soluciones alternativas (refactoring).
- Utilizar la biblioteca pandas junto con matplotlib y numpy para realizar análisis estadísticos de datos y gráficos.

- Como resultado práctico al final del curso cada estudiante habrá creado una aplicación web utilizando el microframework de Python Flask.

1.3 Temario

- Introducción, instalación y compilación
- Datos, expresiones y sentencias
- Variables y funciones, control de flujo
- Clases y objetos, herencia, polimorfismo
- Entradas y salidas con Python
- Gestión de módulos, paquetes y bibliotecas
- Servicios y programas en red, REST API
- Desarrollo de aplicaciones web con Flask
- Análisis de datos con pandas, matplotlib y numpy

- Módulos opcionales
  - SQL ejemplos con pandas
  - Plotting con Python
  - Machine Learning con Python
  - DevOps con Ansible
  - SDK de GCP para Python
  - Kubernetes con Python y Docker

1.4 Bibliografía

- Downey, A., Elkner, J., Meyers, C. Aprenda a Pensar Como un Programador con Python. (2015).
  Recuperado de https://argentinaenpython.com/quiero-aprender-python/aprenda-a-pensar-como-un-programador-con-python.pdf
- Kent D. Lee. Python, Programming Fundamentals Second Edition. 2014.

- Marzal Varó, A., Gracia Luengo, I., García Sevilla, Pedro. Introducción a la programación con Python 3. (2014).

Recuperado de http://repositori.uji.es/xmlui/handle/10234/102653

- Miller, B., Ranum, D. Solución de problemas con algoritmos y estructuras de datos usando Python. Traducido por Mauricio Orozco-Alzate, Universidad Nacional de Colombia - Sede Manizales.
  Recuperado de https://runestone.academy/ns/books/published/pythoned/index.html?mode=browsing

- Shaw, Z. A., Learn Python 3 the Hard Way. (2016).
  Recuperado de https://learnpythonthehardway.org/

- Van Rossum, G. and the Python development team. Documentación de Python en español. (2020).
  Recuperado de https://python-docs-es.readthedocs.io/es/3.10/

2 Servicios y programas en red

- Manipular ficheros, Analizador sintáctico - parser, Formato JSON, Objetos en Python y su equivalente en JSON, Persistencia de datos, Manipular ficheros XML, Seguridad XML, REST API, Herramientas útiles, Descargar ficheros desde Internet, Biblioteca Requests, Beautiful-Soup, Scrapy, Selenium.
- Errores y excepciones en Python: Errores de sintaxis, Excepciones, Depuración de errores, Pruebas unitarias.

2.1 Manipular ficheros

2.2 Analizador sintáctico - parser

- *Parsing* (analizar) en el área de programación significa hacer que los datos se puedan entender, es decir, convertir esos datos a un formato en el cual se pueda trabajar con ellos:
  - Divide cadenas de texto,
  - Las separa con comas o tabuladores,
  - Cambia el formato al fichero,
  - Lo convierte en binario, etc.
- Un ejemplo básico sería convertir un fichero con formato *.csv* a *.ods* para poder trabajar con un programa de ofimática (hoja de cálculo en este caso).
- Según la Wikipedia parser se puede traducir como analizador sintáctico:
  https://es.wikipedia.org/wiki/Analizador_sint%C3%A1ctico.
- En Python tenemos tres posibilidades: usar una biblioteca, desarrollar un *parser* propio, o utilizar una herramienta que genere un *parser*.
- En el primer caso, existen bibliotecas para los formatos de ficheros más conocidos como, XML, HTML, CSV, etc.
- El segundo caso es cuando se necesita mejorar el desempeño, cuando se desea integrar distintos componentes del sistema, o cuando no existe un *parser* para un formato específico de ficheros.
- El tercer caso finalmente sería una vía de medio para los otros dos.

- En los siguientes apartados veremos como analizar ficheros con formatos JSON, XML, y obtener datos de ficheros con logs.
- Tutoriales con ejemplos: https://www.vipinajayakumar.com/parsing-text-with-python/ y https://tomassetti.me/parsing-in-python/.

2.3 Formato JSON

- *JSON (JavaScript Object Notation)* es un formato popular utilizado para representar datos estructurados.
- En Python se utiliza comúnmente para transmitir y recibir datos entre una aplicación y un servidor web.
- El módulo para este fin se denomina *json*.
- El módulo *json* codifica objetos de Python como cadenas JSON y decodifica las cadenas JSON convirtiéndolas en objetos de Python:
  - *json.dumps()*: Obtiene un objeto de Python y lo convierte (*dumps*) a una cadena (*string*).
  - *json.loads()*: Obtiene una cadena JSON y la convierte (*loads*) a un objeto de Python.

[ ]:
```
import json


# Ejemplo de diccionario
persona = {"nombre": "Juan", "lenguajes": ["Python", "Shellscripts"]}
print(persona)
print("Tipo:", type(persona))
```

[ ]:
```
# Ejemplo de string en JSON
persona = '{"nombre": "Juan", "lenguajes": ["Python", "Shellscripts"]}'
print(persona)
print("Tipo:", type(persona))
```

[ ]:
```
# Parsing usando el método json.loads()
# Crea un diccionario en python
persona_dic = json.loads(persona)
print(persona_dic['lenguajes'])
print("Tipo:", type(persona_dic))
```

[ ]:
```
# Convierte diccionario a JSON
# .dumps (encode) toma un diccionario como input y devuelve un string
persona_json = json.dumps(persona_dic)
print(persona_json)
print("Tipo:", type(persona_json))
```

- Para leer un fichero *.json* y luego exportarlo:

[ ]:
```
import json
```

```python
with open('ejemplo_json_fichero.json') as fichero:
    # loads (decode) toma un string como input y devuelve un diccionario
    datos = json.loads(fichero.read())
    print(datos)

# Lee solo un dato del fichero
print(datos['Nombre'])

# Exporta la variable datos a un fichero .json
with open('ejemplo_persona_json.txt', 'w') as json_fichero:
    json.dump(datos, json_fichero)
```

2.4 Objetos en Python y su equivalente en JSON

Python	JSON
dict	object
list, tuple	array
str	string
int, float, int	number
True	true
False	false
None	null

- Fuente: https://rico-schmidt.name/pymotw-3/json/index.html.

2.5 Persistencia de datos

- Los módulos en Python que soportan el almacenamiento de datos de forma persistente en el disco se denominan *pickle* y *marshal*.
- De los dos *pickle* es el recomendado, además de ser el más utilizado.
- Éstos pueden convertir varios tipos de datos de Python en un flujo de bytes y luego recrear los objetos a partir de los mismos.
- Los módulos relacionados con bases de datos en cambio admiten formatos de archivo basados en *hash* que almacenan un mapeo de cadenas a otras cadenas.

- Existe también un módulo llamado *jsonpickle* que permite codificar y decodificar objetos en Python.
- Se recomienda usarlo con cautela ya que puede presentar problemas de seguridad al ejecutar código arbitrario.
- https://github.com/jsonpickle/jsonpickle.

- "Serializar" (*pickling*) significa convertir una jerarquía de objetos de Python en una secuencia de bytes,
- "Deserializar" (*unpickling*) es la operación inversa.

- Tener en cuenta que el módulo *pickle* no es seguro, solo se deben "deserializar" datos en los cuales se confía, para datos no seguros el uso de JSON es más apropiado. Fuente: https://docs.python.org/es/3/library/pickle.html.

- Algunas diferencias entre JSON y *pickle*:
  - JSON codifica a UTF-8, es legible para seres humanos, interoperable, y representa solo un subconjunto de los tipos integrados de Python.
  - *Pickle* por su parte convierte a código binario, no es legible para el ser humano, es específico de Python y puede representar un número muy grande de tipos de Python.

[ ]:
```
import pickle


datos = int(input('Cantidad de datos a ingresar: '))
lista_datos = []

for dato in range(datos):
    datos_in = input('Ingresar dato '+str(dato)+' : ')
    lista_datos.append(datos_in)


fichero = open('pickle_datos', 'wb')
pickle.dump(lista_datos, fichero)
fichero.close()
```

[ ]:
```
import pickle


fichero = open('pickle_datos', 'rb')

datos = pickle.load(fichero)
fichero.close()

print('Datos guardados en el fichero:')

lista = 0
for item in datos:
    print('El dato ', lista, ' es : ', item)
    lista += 1
```

2.6 Manipular ficheros XML

- *XML (eXtensible Markup Language)* es un lenguaje ampliamente utilizado para almacenar y transportar datos estructurados.
- En Python se utilizan la biblioteca *xml.etree.ElementTree - The ElementTree XML API*, o la biblioteca *minidom*, aunque la primera es aconsejable si no se conoce la API del DOM.
- *Minidom (Minimal DOM Implementation)* es una implementación simplificada del *Document Object Model (DOM)*.
- La API del DOM trata al XML como una estructura de árbol donde cada rama o nodo es un objeto.
- https://developer.mozilla.org/es/docs/Web/API/Document_Object_Model/

Introduction.

- https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model.
- El siguiente ejemplo de uso de la biblioteca *xml.etree.ElementTree* fue extraído y adaptado de la documentación oficial https://docs.python.org/3.9/library/xml.etree.elementtree.html#module-xml.etree.ElementTree.

[ ]:
```
import xml.etree.ElementTree as ET
# Importar XML desde un fichero
arbol = ET.parse('ejemplo_country_data.xml')
raiz = arbol.getroot()

# Acceder a elementos determinados
for rama in raiz:
    print(rama.tag, rama.attrib)
# Acceder al child utilizando índices
print(raiz[0][1].text)
```

[ ]:
```
# Acceder a sub elemento utilizando el método Element
for neighbor in raiz.iter('neighbor'):
    print(neighbor.attrib)
```

[ ]:
```
# findall encuentra sólo los elementos con un tag que son hijos directos del␣
↪elemento actual
for country in raiz.findall('country'):
    rank = country.find('rank').text
    name = country.get('name')
    print(name, rank)
```

[ ]:
```
# Modificar un fichero XML y escribirlo a un nuevo fichero
for rank in raiz.iter('rank'):
    new_rank = int(rank.text) + 1
    rank.text = str(new_rank)
    rank.set('Actualizado', 'si')
arbol.write('ejemplo_country_data_modificado.xml')
```

2.7 Seguridad XML

- Ataques a bibliotecas XML pueden ser extremadamente dañinos.
- Los atacantes suelen utilizar funcionalidades poco conocidas de XML.
- La biblioteca *defusedxml* provee varias medidas de contorno para rechazar estos ataques.

[ ]:
```
# Instalación
pip install defusedxml
```

- Para utilizarla:

[ ]:
```
# Cambiar este código
from xml.etree.ElementTree import parse
et = parse(xmlfile)

# Por este otro:
from defusedxml.ElementTree import parse
et = parse(xmlfile)
```

- Fuente: https://pypi.org/project/defusedxml/#python-xml-libraries

2.8 REST API

- *REST (REpresentational State Transfer Application Programming Interface)* o API de *RESTful* es en esencia una serie de convenciones que sirven para estructurar una API que interactúa utilizando el protocolo HTTP.
- Permite hacer pedidos a una URL mediante dos métodos: GET, que obtiene datos, y POST, que los envía.
- El ejemplo clásico es obtener un formulario (GET) de una página web y una vez relleno, enviarlo (POST).
- Otros métodos son PUT (reemplaza datos) y DELETE (borra datos), aunque hay más.

- Python cuenta con la biblioteca *urllib.request* que define funciones y clases para:
  - apertura de URLs mediante HTTP, autenticación básica y digest, redirecciones, cookies y más.
- El listado de todos los métodos existentes se puede consultar en la documentación oficial.
- Fuente: https://docs.python.org/es/3/library/urllib.request.html#module-urllib.request.

- Las APIs como *REST* utilizan el lenguaje *JSON (JavaScript Object Notation)* que codifica las estructuras de datos de manera tal que sean legibles por las máquinas y puedan ser enviados y recibidos.
- En Python se utilizan mayoritariamente las bibliotecas *requests* para usar *REST* y *json* para convertir datos en ese formato a diccionarios o listas.

[ ]:
```
# Instalar la biblioteca Requests
pip install requests
```

- Un ejemplo de uso de *REST* adaptado del tutorial de https://www.dataquest.io/blog/python-api-tutorial/.

[ ]:
```
import requests
import json
from datetime import datetime

# Módulo REQUESTS
# Ejemplo utilizando la API de Open Notify http://api.open-notify.org/
# Utiliza la función request.get() para conectarse a una API inexistente
response = requests.get("http://api.open-notify.org/esta-api-no-existe")
print(response.status_code)
```

```
[ ]: # Función request.get() con código de estado 200
response = requests.get("http://api.open-notify.org/astros.json")
print(response.status_code)
```

```
[ ]: # Función response.json() devuelve datos utilizando la API
print(response.json())
print(type(response.json()))
```

```
[ ]: # Módulo JSON
# Función creada para dar formato a los datos
def json_print(obj):
    texto = json.dumps(obj, sort_keys=True, indent=4)
    print(texto)
json_print(response.json())
```

```
[ ]: # Obtener datos de iss-now.json
respuesta = requests.get("http://api.open-notify.org/iss-now.json")
print(respuesta.status_code)


def json_print(objeto):
    texto = json.dumps(objeto, sort_keys=True, indent=4)
    print(texto)


json_print(respuesta.json())
```

```
[ ]: # Módulo DATETIME
# Extrae los datos de "timestamp"
# Presenta los datos con formato fecha
fecha_unix = respuesta.json()["timestamp"]
json_print(f'Fecha formato Unix: {fecha_unix}')

fecha = datetime.fromtimestamp(fecha_unix)
json_print(f'Fecha formato normal: {fecha}')
```

2.9 Herramientas útiles

- Listado de APIs públicas: https://github.com/public-apis/public-apis.
- Listado Rapidapi: https://rapidapi.com/collection/list-of-free-apis.
- API markerplace: https://apilayer.com/.
- Un simple servicio para HTTP Request & Response: https://httpbin.org/.
- Clientes REST/API:
  - https://insomnia.rest/ Insomnia,
  - https://www.paw.cloud/ Paw, y
  - https://www.postman.com/ Postman.
- Para conocer los códigos de errores HTTP se puede visitar la siguiente página web de la

Wikipedia:
https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP.

2.10 Descargar ficheros desde Internet

- Para descargar ficheros con Python desde Internet se pueden utilizar entre otras, las bibliotecas *urllib.request*, *wget*, *requests*, y *urllib3*.
- La biblioteca estándar de Python para estas tareas es *urllib* aunque se utilizan mayoritariamente *requests* junto con *urllib3*.
- La biblioteca *urllib* es una colección de varios modulos que permiten trabajar con URLs:
  - *urllib.request* Para abrir y leer URLs.
  - *urllib.error* para gestionar las excepciones recogidas por *urllib.request*.
  - *urllib.parse* para parsear URLs.
  - *urllib.robotparser* para parsear ficheros *robots.txt*.
- Fuente: https://docs.python.org/3/library/urllib.request.html#module-urllib.request

[ ]:
```
# Ejemplo descargar y descomprimir un fichero con urllib.request
# Fichero con los datos de https://insights.stackoverflow.com/survey
from io import BytesIO
from urllib.request import urlopen
from zipfile import ZipFile
```

[ ]:
```
zip_url = 'https://info.stackoverflowsolutions.com/rs/719-EMH-566/images/
 ↪stack-overflow-developer-survey-2021.zip'
```

[ ]:
```
with urlopen(zip_url) as zipresp:
    with ZipFile(BytesIO(zipresp.read())) as zfile:
        zfile.extractall('/home/usuario/')    # Directorio donde guardar los␣
 ↪ficheros
```

- *Wget* es una herramienta de los sistemas GNU/Linux y Unix que permite descargar ficheros de Internet.
- El módulo de Python con el mismo nombre se instala con:

[ ]:
```
pip install wget
```

[ ]:
```
# Ejemplo con wget
import wget


url='https://info.stackoverflowsolutions.com/rs/719-EMH-566/images/
 ↪stack-overflow-developer-survey-2021.zip'
wget.download(url)
```

2.11 Biblioteca requests

- La biblioteca *requests* es la recomendada para una interfaz de cliente HTTP de mayor nivel.
- Permite enviar pedidos HTTP/1.1 de manera sencilla.
- Utiliza *urllib3* para mantener activas las conexiones HTTP de forma automática.
- Fuente: https://requests.readthedocs.io/en/latest/.

[ ]:
```
# Instalación
python -m pip install requests
```

[ ]:
```
# Ejemplo descargar y descomprimir un fichero con requests
import requests


url = 'https://info.stackoverflowsolutions.com/rs/719-EMH-566/images/
 ↪stack-overflow-developer-survey-2021.zip'
req = requests.get(url)

# Se obtiene solamente el nombre del fichero en la URL
fichero = url.split('/')[-1]

with open(fichero, 'wb') as output_file:
    output_file.write(req.content)
```

- *urllib3* proporciona varias características no disponibles en la biblioteca estándar de Python, entre ellas:
  - Seguridad.
  - *Pool* de conexiones.
  - Verificación SSL/TLS del lado del cliente.
  - Cargas de ficheros con codificación multiparte.
  - Asistentes para reintentar solicitudes y gestionar redireccionamientos HTTP.
  - Compatibilidad con la codificación gzip, deflate, brotli y zstd.
  - Compatibilidad con proxy para HTTP y SOCKS.
- Fuente: https://github.com/urllib3/urllib3

[ ]:
```
# Instalación
python -m pip install urllib3
```

2.12 BeautifulSoup

- *BeautifulSoup* es un parser o analizador sintáctico HTML ideal para realizar *web scraping*.
- Permite abrir páginas web y extraer datos estructurados a partir de patrones para luego manipularlos.
- El web scraping con BeautifulSoup tiene la siguiente estructura:
  - URL → Solicitud HTTP → HTML → BeautifulSoup

[ ]:
```
pip install beautifulsoup4 # Instalación de la biblioteca
pip install lxml # Instalación del parser a utilizar con la biblioteca
```

[ ]:
```
# Ejemplo uso de la biblioteca Beautiful Soup
# Documentación oficial:
# https://www.crummy.com/software/BeautifulSoup/bs4/doc/
from bs4 import BeautifulSoup
import requests


url = input("Escribe la URL del sitio web: ")
agrega_http = requests.get("http://" + url)
datos = agrega_http.text

soup = BeautifulSoup(datos, features="html5lib")

for link in soup.find_all("a"):
    print(link.get("href"))
```

[ ]:
```
# Ejemplo extraído de https://scipython.com/blog/
 ↪scraping-a-wikipedia-table-with-beautiful-soup/
import urllib.request
from bs4 import BeautifulSoup


url = 'https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2'
req = urllib.request.urlopen(url)
article = req.read().decode()

with open('ISO_3166-1_alpha-2.html', 'w') as fo:
    fo.write(article)

# Load article, turn into soup and get the <table>s.
article = open('ISO_3166-1_alpha-2.html', encoding='UTF-8').read()
soup = BeautifulSoup(article, 'html.parser')
tables = soup.find_all('table', class_='sortable')

# Search through the tables for the one with the headings we want.
for table in tables:
    ths = table.find_all('th')
    headings = [th.text.strip() for th in ths]
    if headings[:5] == ['Code', 'Country name', 'Year', 'ccTLD', 'ISO 3166-2']:
        break

# Extract the columns we want and write to a semicolon-delimited text file.
with open('iso_3166-1_alpha-2_codes.txt', 'w') as fo:
    for tr in table.find_all('tr'):
```

```
        tds = tr.find_all('td')
        if not tds:
            continue
        code, country, year, ccTLD = [td.text.strip() for td in tds[:4]]
        # Wikipedia does something funny with country names containing
        # accented characters: extract the correct string form.
        if '!' in country:
            country = country[country.index('!')+1:]
        print('; '.join([code, country, year, ccTLD]), file=fo)
```

2.13 Scrapy

- Otra herramienta de scraping para Python es Scrapy https://scrapy.org/:
  - Servicio https://www.zyte.com/scrapy-cloud/
  - Scrapy spiders https://scrapyd.readthedocs.io/en/stable/

[ ]:
```
pip install scrapy
```

[ ]:
```
# Ejemplo extraído de https://scrapy.org/
import scrapy

class BlogSpider(scrapy.Spider):
    name = 'blogspider'
    start_urls = ['https://www.zyte.com/blog/']

    def parse(self, response):
        for title in response.css('.oxy-post-title'):
            yield {'title': title.css('::text').get()}

        for next_page in response.css('a.next'):
            yield response.follow(next_page, self.parse)
```

[ ]:
```
scrapy runspider myspider.py
```

2.14 Selenium

- Selenium facilita una API para poder escribir tests de automatización de aplicaciones web.
- También permite automatizar tareas de administración web.
- Tiene soporte para los navegadores más utilizados actualmente como Firefox, Chrome, IE, y Safari.
- Fuente: https://www.selenium.dev/.
- Documentación Selenium: https://selenium-python.readthedocs.io/.
- Se instala con:

[ ]:
```
pip install selenium
```

- Para poder hacer las pruebas se deberán instalar los controladores para los navegadores web.

- https://www.selenium.dev/documentation/webdriver/getting_started/install_drivers/.

[ ]:
```
# Ejemplo: Se instala el controlador para Firefox en sistemas GNU/Linux Debian/
↪Ubuntu
sudo apt install firefox-geckodriver
```

[ ]:
```
# Ejemplo básico de automatización
from selenium import webdriver


probar = webdriver.Firefox()
probar.get('https://fortinux.com')
print('Título de la página: %s', probar.title)
print(probar.page_source)
probar.quit()
```

2.15 Errores y excepciones en Python

- En python dentro de los diferentes tipos de errores se pueden mencionar los errores de sintaxis y las excepciones.
- Todas las excepciones deben ser instancias de una clase que se derive de *BaseException*.
- Las nuevas excepciones se recomienda que deriven a partir de la clase *Exception*.
- Las excepciones en Python se manejan esencialmente con *try-except*.
- Documentación oficial: https://docs.python.org/es/3/tutorial/errors.html.
- Excepciones incorporadas: https://docs.python.org/es/3/library/exceptions.html.
- Definidas por el usuario https://docs.python.org/es/3/tutorial/errors.html#tut-userexceptions.

2.16 Errores de sintaxis

- Los errores de sintaxis se muestran cuando Python no consigue interpretar el código debido a una sintaxis erronea.

[2]:
```
>>> print ('Hola mundo)
  File "<stdin>", line 1
    print ('Hola mundo)
```

```
  File "/tmp/ipykernel_783/2391534570.py", line 1
    print ('Hola mundo)
                       ^
SyntaxError: EOL while scanning string literal
```

2.17 Excepciones

- Los errores en Python detectados durante la ejecución se llaman excepciones.

- El bloque *try* prueba un bloque de código en busca de errores.
- El bloque *except* gestiona los errores.
- El bloque *finally* permite ejecutar código, a pesar de los resultados de *try* - *except*.

[ ]:
```
def funcion_excepciones():
    try:
        # Dividir entre cero genera una excepción
        print(10 / 0)
    except ZeroDivisionError:
        print("Error. No se puede dividir por cero.")
    else:
        # La excepción no ha ocurrido
        print("La excepción no ha ocurrido")
    finally:
        # Este bloque se ejecuta cuando todas
        # las excepciones han sido ejecutadas
        print("Finalizadas las excepciones.")


funcion_excepciones()
```

- ZeroDivisionError: División en la que el divisor es cero o no se puede distinguir de cero.
- ValueError: Valores inapropiados. Cuando una función (como *int()* o *float()*) recibe un argumento de un tipo adecuado, pero su valor es inaceptable.
- TypeError: Dato con un tipo inadecuado.

[1]:
```
lista = [5]
valor = lista[0.5]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_3587/2039152235.py in <module>
      1 # TypeError: Dato con un tipo inadecuado
      2 lista = [5]
----> 3 valor = lista[0.5]

TypeError: list indices must be integers or slices, not float
```

- SyntaxError: Cuando una línea de código viola la gramática de Python.

[2]:
```
def funcion()
    return
```

```
  File "/tmp/ipykernel_3587/2294903513.py", line 1
    def funcion()
                 ^
```

```
SyntaxError: invalid syntax
```

- AttributeError: Cuando se intenta utilizar un método que no existe para ese elemento.

[9]:
```
lista = [5]
lista.depend(2)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/ipykernel_770/636947093.py in <module>
      1 lista = [5]
----> 2 lista.depend(2)

AttributeError: 'list' object has no attribute 'depend'
```

- NameError: Cuando no se encuentra un nombre local o global.

[2]:
```
5 * elemento
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_452/2474722050.py in <module>
----> 1 5 * elemento

NameError: name 'elemento' is not defined
```

- Por último *raise* permite forzar a que ocurra una excepción:

[3]:
```
>>> raise NameError("Hola Mundo")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: Hola Mundo
```

```
  File "/tmp/ipykernel_3587/3742110996.py", line 2
    Traceback (most recent call last):
                      ^
SyntaxError: invalid syntax
```

[3]:
```
variable = "1"

if not type(variable) is int:
  raise TypeError("Solamente se admiten números enteros")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_452/1387624469.py in <module>
      2
      3 if not type(variable) is int:
----> 4   raise TypeError("Solamente se admiten números enteros")

TypeError: Solamente se admiten números enteros
```

2.18 Depuración de errores

- El proceso durante el cual se eliminan los errores del código se llama depuración.
- Una técnica de depuración simple es la conocida como la depuración por impresión:
  - Se inserta *print()* dentro del código para verificarlo
  - Por ej. imprimir valores de variables.
- Otra técnica es el uso de *assertions* en Python:

[5]:
```
import sys
assert ('linux' in sys.platform), "solamente se ejecuta en sistemas Linux"
assert ('win32' in sys.platform), "solamente se ejecuta en sistemas Windows"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
/tmp/ipykernel_452/3267745790.py in <module>
      1 import sys
      2 assert ('linux' in sys.platform), "solamente se ejecuta en sistemas␣
 ↪Linux"
----> 3 assert ('win32' in sys.platform), "solamente se ejecuta en sistemas␣
 ↪Windows"

AssertionError: solamente se ejecuta en sistemas Windows
```

2.19 Pruebas unitarias

- Otra técnica es la llamada de pruebas unitarias.
- Python proporciona un módulo llamado *unittest*. https://docs.python.org/es/3.10/library/unittest.html
- El framework *pytest* https://docs.pytest.org/en/latest/ hace fácil escribir pequeñas pruebas, además de poder escalar y soportar complejas pruebas funcionales para aplicaciones y bibliotecas.
- Un tutorial se puede encontrar en https://fortinux.com/tutoriales/realizar-pruebas-codigo-python/