Python

Instructor:
L.I. Arturo Rendón Cruz

Contenido

- Introducción
  - ¿Qué es Python?
  - Características de Python
  - ¿Por qué Python?
- Conceptos básicos de Python
  - Tipos básicos
  - Colecciones
  - Sentencias condicionales
- Funciones
- Orientación a objetos
- Python Enhancement Proposals (PEP)
- Frameworks para el desarrollo Web

Introducción

¿Qué es Python?

- Es un lenguaje de programación creado por Guido van Rossum a principios de los 90’s cuyo nombre esta inspirado en el grupo de cómicos ingleses “Monty Python”.
- Es un lenguaje interpretado o de script ya que se ejecuta ya que se ejecuta utilizando un interprete en lugar de compilar el código.

Características de Python

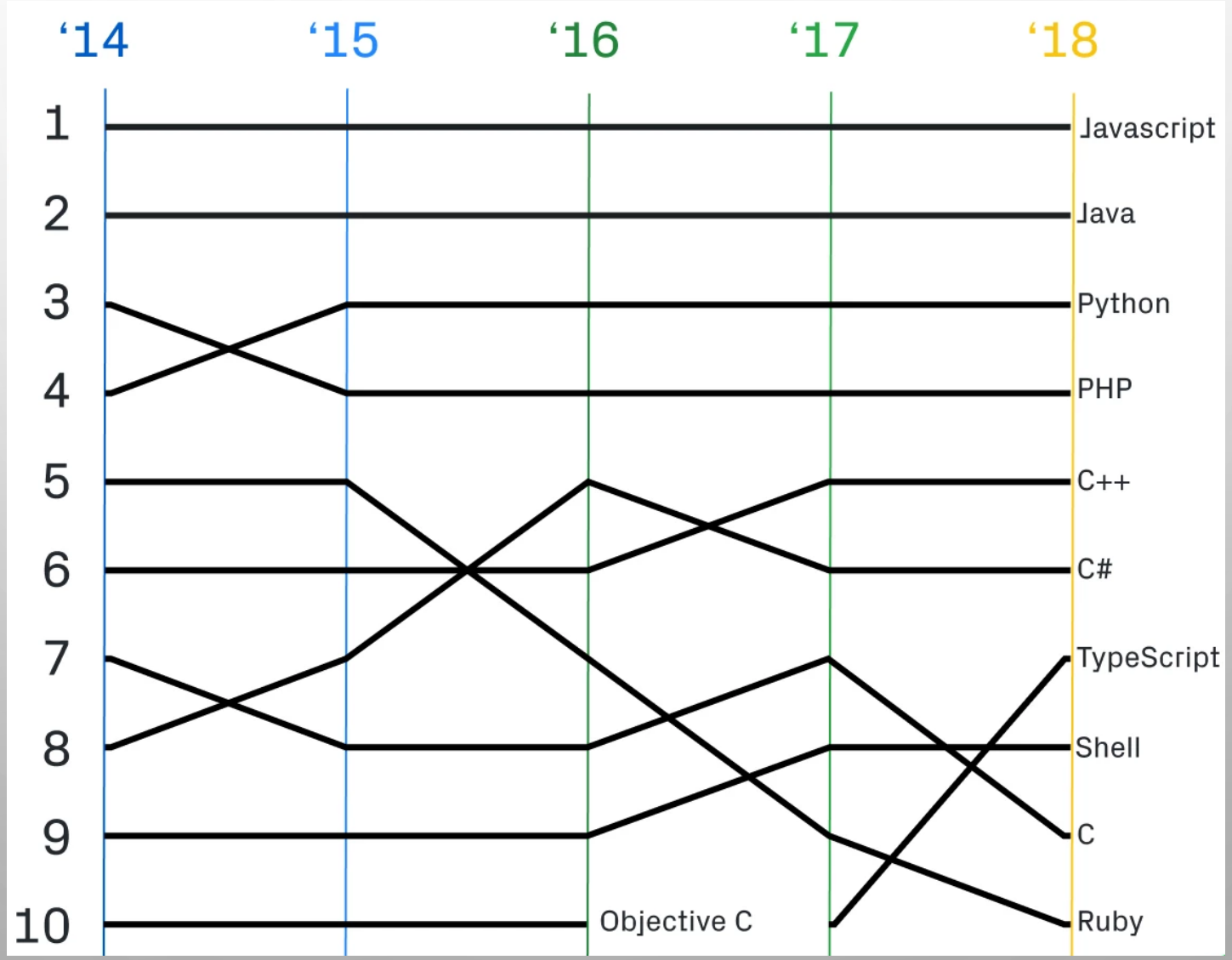
- Tipado dinámico: El tipo de dato de una variable se determina según el valor asignado.
- Fuertemente tipado: No se pueden tratar a variables como otro tipo distinto al definido originalmente.
- Multiplataforma: Python puede ejecutarse en UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.
- Orientado a objetos

¿Por qué Python?

“Su sintaxis simple, clara y sencilla; el tipado dinámico, el gestor de memoria, la gran cantidad de bibliotecas disponibles y la potencia del lenguaje, entre otros, hacen que desarrollar una aplicación en Python sea sencillo, muy rápido” (Gonzáles Duque, Raul)

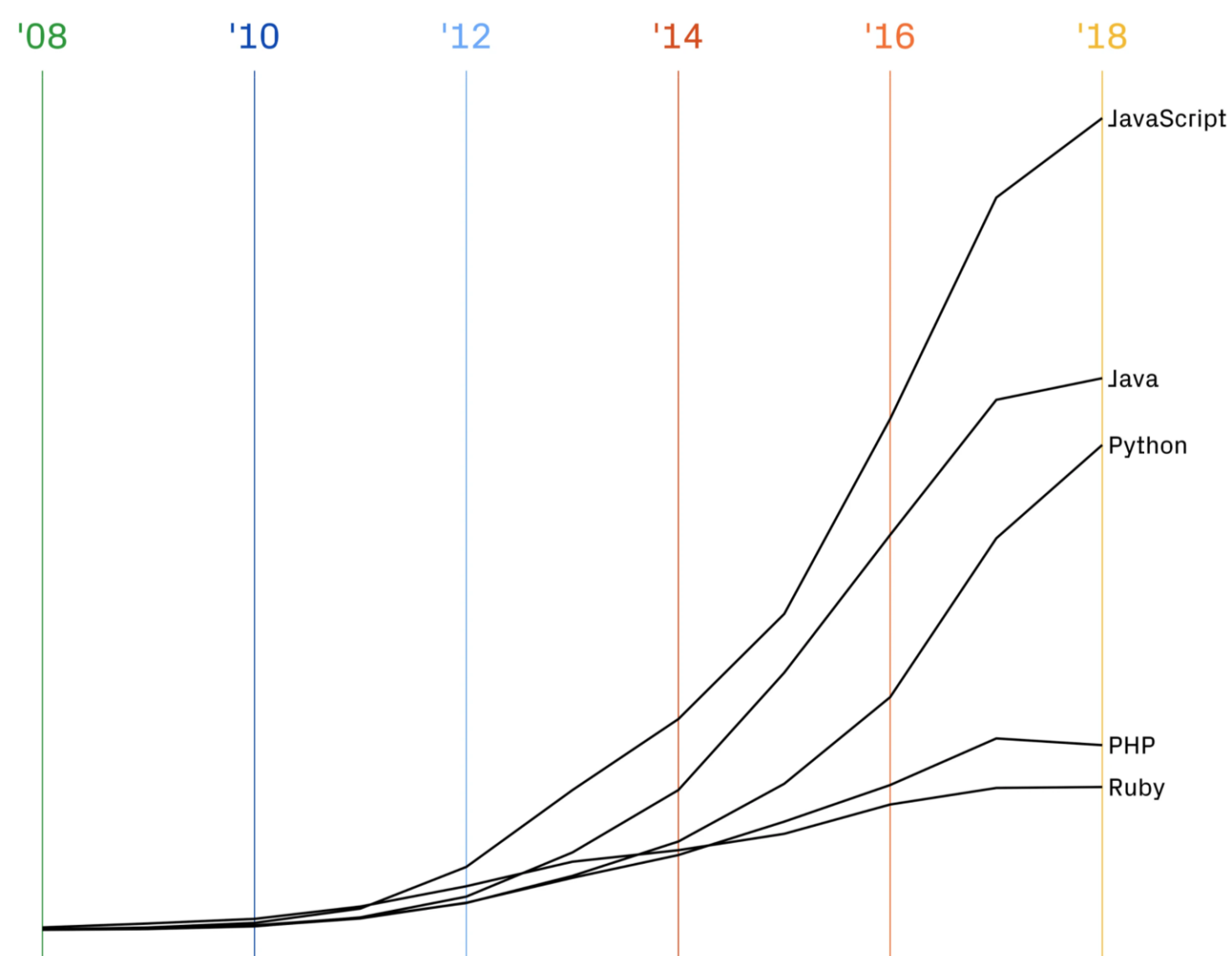
¿Por qué Python?

https://bit.ly/2OcY45U



¿Por qué Python?

https://bit.ly/2OcY45U



Conceptos de Python
Tipos básicos: Números

- Enteros: Número positivo o negativo sin puntos decimales y se expresan con el tipo *int*

```
>>> entero = 10
>>> type(entero)
<class 'int'>
```

```
>>> entero_hex = 0xA
>>> type(entero_hex)
<class 'int'>
>>> entero_hex
10
```

```
>>> entero_oct = 0o12
>>> type(entero_oct)
<class 'int'>
>>> entero_oct
10
```

Conceptos de Python
Tipos básicos: Números

- Reales: Número positivo o negativo con puntos decimales y se expresan con el tipo *float*

```
>>> pi = 3.141592653589793
>>> type(pi)
<class 'float'>
```

Conceptos de Python
Tipos básicos: Números

- Complejos: Los números complejos son aquellos que tienen parte imaginaria.

```
>>> complejo =  2.1 + 7.8j
>>> type(complejo)
<class 'complex'>
>>> complejo
(2.1+7.8j)
```

Conceptos de Python
Tipos básicos: Números

- Complejos: Los números complejos son aquellos que tienen parte imaginaria.

```
>>> complejo =  2.1 + 7.8j
>>> type(complejo)
<class 'complex'>
>>> complejo
(2.1+7.8j)
```

Conceptos de Python

Tipos básicos: Números

Operadores aritméticos

- Suma +
- Resta -
- Multiplicación *
- Exponente **
- División /
- División entera //
- Módulo %

Conceptos de Python
Tipos básicos: Números

Operadores booleanos

- and &
- or |
- xor ^
- not ~
- Desplazamiento a la derecha >>
- Desplazamiento a la izquierda <<

Conceptos básicos de Python

Tipos básicos: Cadenas

Las cadenas de texto o string puden ser representadas encerrando el contenido dentro de comillas simples o dobles.

```
>>> var = 'hola mundo!'
>>> type(var)
<class 'str'>
>>> var
'hola mundo!'
```

Prefijos

- u: Se utiliza al definir lena Unicode
- r: Para definir cadenas raw

https://www.python.org/dev/peps/pep-0008/#string-quotes
https://www.python.org/dev/peps/pep-0414/

Conceptos básicos de Python

Tipos básicos: Booleanos

El tipo de dato *bool* es una subclase del tipo *int* que solo pude tener dos valores *True* o *False*

```
>>> falso = False
>>> type(falso)
<class 'bool'>
>>> verdadero = True
>>> type(verdadero)
<class 'bool'>
```

Conceptos básicos de Python

Tipos básicos: Booleanos

Operadores lógicos o condicionales

- and: Es verdadero cuando todos los valores son *True*

- or: Es verdadero cuando alguno de los valores es *True*
- not: Niega el valor de booleano

Conceptos básicos de Python

Tipos básicos: Booleanos

Valores booleanos como resultado de operaciones relacionales

```
>>> 2 == 2
True
>>> 2 != 3
True
>>> 2 < 3
True
>>> 2 > 3
False
>>> 3 <= 3
True
>>> 3 >= 3
True
```

Operador	Des	
==	Igua	2 == 2
!=	Dife	2 != 3
<	Mer	2 < 3
>	May	2 > 3
<=	Mer	3 <= 3
>=	May	3 >= 3

Conceptos de Python
Colecciones: Listas

- La lista es un tipo de colección ordenada que pude contener números, cadenas, booleanos u otras listas.
- Se puede definir una lista indicando los valores entre corchetes separados por comas.

```
>>> lista = [22, True, "hola mundo", [1, 2]]
>>> type(lista)
<class 'list'>
```

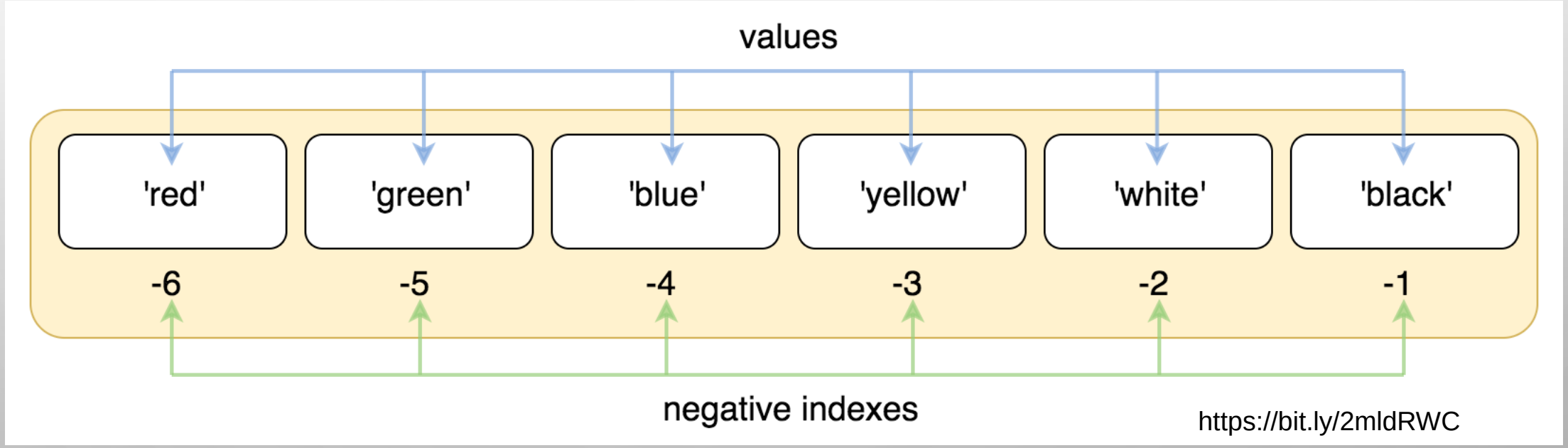
Conceptos de Python
Colecciones: Listas

- Se accede a los valores de una lista por medio del índice entre corchetes.

```
>>> lista[0]
22
```

```
>>> lista[3][1]
2
```

Conceptos de Python
Colecciones: Listas
Índices negativos



Conceptos de Python
Colecciones: Listas

Slicing

El *slicing* o particionado permite obtener subconjuntos de una lista utilizando la siguiente notación dentro de los corchetes:

- [inicio:fin]: para obtener un sub conjunto desde la inicio(inclusivo) hasta el fin(no inclusivo).

```
>>> lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> lista[3:6]
[4, 5, 6]
```

Conceptos de Python
Colecciones: Listas

Slicing

- [inicio:fin:salto]: para obtener un sub conjunto desde la inicio(inclusivo) hasta el fin(no inclusivo), el salto se utiliza para tomar valores de la lista evitando los que se encentren en medio del salto.

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> lista[0:10:2]
[1, 3, 5, 7, 9]
```

Conceptos de Python
Colecciones: Listas
Slicing

- [:fin]: Se pueden obtener los primeros valores de una lista omitiendo el índice 0 esta notación es igual a [0:fin]

```
>>> lista[0:5]
[1, 2, 3, 4, 5]
>>> lista[:5]
[1, 2, 3, 4, 5]
```

Conceptos de Python
Colecciones: Listas
Slicing

- [incio:]: De igual forma se pueden obtener los últimos valores de una lista omitiendo el final de la misma.

```
>>> lista[:5]
[1, 2, 3, 4, 5]
>>> lista[5:]
[6, 7, 8, 9, 10]
>>> lista[-5:]
[6, 7, 8, 9, 10]
```

Conceptos de Python
Colecciones: Listas

Slicing

- Operaciones útiles con las listas.
- [::2]: Posiciones pares de la lista

```
>>> lista[::2]
[1, 3, 5, 7, 9]
```

- [1::2]: Posiciones impares de la lista

```
>>> lista[1::2]
[2, 4, 6, 8, 10]
```

Conceptos de Python
Colecciones: Listas
Slicing

- [::-1]: Lista inversa

```
>>> lista[::-1]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Conceptos de Python

Colecciones: Tuplas

Las tuplas se definen con separando únicamente los valores con comas, se puede utilizar el paréntesis en lugar de corchete para dar claridad a la definición de las tuplas.

```
>>> tupla[0] = 20
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Las tuplas tienen las mismas operaciones que listas a diferencia de la modificación ya que es un objeto inmutable.

Conceptos de Python

Colecciones: Diccionarios

Los diccionarios o matrices asociativas son colecciones que relacionan una clave con un valor, los diccionarios no tienen un orden y se accede a los valores por por medio de la clave entre corchetes.

```
>>> diccionario = {
...     'clave': 'valor',
...     'uno': 1,
...     'dos': 2
... }
>>> type(diccionario)
<class 'dict'>
```

Conceptos de Python

Sentencias condicionales

La forma más simple de una sentencia condicional es un *if* que evaluará una operación relacional para ejecutar un código en caso de que el resultado se *True.*

```
>>> var = 10
>>> if var == 10:
...     print("El valor es 10")
...
El valor es 10
```

Conceptos de Python
Sentencias condicionales

Cuando la condición no se debemos usar la sentencia condicional *else*

```
>>> var = 9
>>> if var == 10:
...     print("El valor es 10")
... else:
...     print("El valor es diferente de 10")
...
El valor es diferente de 10
```

Conceptos de Python
Sentencias condicionales

Pueden existir más comparaciones para una sentencia condicional para eso podemos ultilizar *elif*, que es una contracción de *else if*

```
>>> var = 9
>>> if var == 10:
...     print("El valor es 10")
... elif var == 9:
...     print("El valor es 9")
```

Concep

Bucles: w

El bucle o c el el resultado c

```
>>> edad = 0
>>> while edad < 18:
...     edad += 1
...     print("Felicidades, tienes " + str(edad))
...
Felicidades, tienes 1
Felicidades, tienes 2
Felicidades, tienes 3
Felicidades, tienes 4
Felicidades, tienes 5
Felicidades, tienes 6
Felicidades, tienes 7
Felicidades, tienes 8
Felicidades, tienes 9
Felicidades, tienes 10
Felicidades, tienes 11
Felicidades, tienes 12
Felicidades, tienes 13
Felicidades, tienes 14
Felicidades, tienes 15
Felicidades, tienes 16
Felicidades, tienes 17
Felicidades, tienes 18
```

python

Conceptos de Python

Bucles: *for … in*

En Python el bucle *for* se utiliza para facilitar la iteración de elementos de una secuencia.

```
>>> secuencia = ["uno", "dos", "tres"]
>>> for elemento in secuencia:
...     print(elemento)
...
uno
dos
tres
```

Funciones

Una funció serie de tareas y de s procedimie caso de no especificar

```
>>> def suma(parametro1, parametro2):
...     resultado = parametro1 + parametro2
...     print(resultado)
...
>>> suma(4, 3)
7
>>> def suma(parametro1, parametro2):
...     resultado = parametro1 + parametro2
...     return resultado
...
>>> print(suma(4,3))
7
```

python

Orientación a objetos

Python es en leguaje multiparadigma, por lo tanto se puede trabajar con programación estructurada, funcional u orientada a objetos.

En realidad en Python todo es un objeto por lo que el paradigma principal es el orientado a objetos, donde la abstracción de los conceptos de modela a través de clases y objetos, y la interacción de los programas se realiza a través de estos objetos.

Or

En P
clas

```
>>> class Coche:
...     """Constructor de la clase coche."""
...     def __init__(self, gasolina):
...         self.gasolina = gasolina
...         print("Tenemos", gasolina, "litros")
...     def arrancar(self):
...         if self.gasolina > 0:
...             print("Arranca")
...         else:
...             print("No arranca")
...     def conducir(self):
...         if self.gasolina > 0:
...             self.gasolina -= 1
...             print("Quedan", self.gasolina, "litros")
...         else:
...             print("No se mueve")
...
```

https://bit.ly/2mjyMsZ

python

Orientación a objetos

Creación del objeto

```
>>> coche = Coche(4)
Tenemos 4 litros
>>> coche.arrancar()
Arranca
>>> coche.conducir()
Quedan 3 litros
>>> coche.conducir()
Quedan 2 litros
>>> coche.conducir()
Quedan 1 litros
>>> coche.conducir()
Quedan 0 litros
```

Orienta

Herencia

Una de las a a
Objetos es l
(superclase ue
heredará lo es Duque,
Raúl)

```
>>> class Instrumento:
...     def __init__(self, precio):
...         self.precio = precio
...     def tocar(self):
...         print("Estamos tocando musica")
...     def romper(self):
...         print("Eso lo pagas tu")
...         print("Son", self.precio, "$$$")
...
>>> class Bateria(Instrumento):
...     pass
...
>>> class Guitarra(Instrumento):
...     pass
...
```

https://bit.ly/2mjyMsZ

python

Orientación a objetos

Herencia m

Python perm aso de
existir métod teniendo
mayor impor

```
>>> class Terrestre:
...     def desplazar(self):
...         print("El animal anda")
...
>>> class Acuatico:
...     def desplazar(self):
...         print("El animal nada")
...
>>> class Cocodrilo(Terrestre, Acuatico):
...     pass
...
>>> c = Cocodrilo()
>>> c.desplazar()
El animal anda
```

https://bit.ly/2mjyMsZ

python

Orientación a objetos

Polimorfismo

En Python el polimorfismo se lleva a cabo a través de la herencia, dónde el comportamiento puede cambiar al reescribir un método de la superclase en la definición de la subclase, en ocasiones se utiliza al termino polimorfismo a la sobrecarga de métodos, Pyhon al ser un lenguaje de tipado dinámico no impone restricciones a los parámetros que se pasan a una función, por lo que este tipo de comportamiento no es relevante.

Orientación a objetos

```
>>> class Ejemplo:
...     def publico(self):
...         print("Publico")
...     def __privado(self):
...         print("Privado")
...
>>> ej = Ejemplo()
>>> ej.publico()
Publico
>>> ej.__privado()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Ejemplo' object has no attribute '__privado'
```

https://bit.ly/2mjyMsZ

Orientación a objetos

Encapsulamiento

La encapsulación en los lenguajes de programación Orientada a Objetos es la capacidad de restringir el acceso a los métodos y atributos de la clase, en leguajes de programación como Java se determina la visibilidad por medio anteponiendo public o private, para el caso de Python todos los elementos son públicos y para definir un elemento privado el nombre del atributo o método debe iniciar con dos guiones bajos (__atributo)

Index by Category

Meta-PEPs (PEPs about PEPs or Processes)

	PEP	PEP Title	PEP Author(s)
P	1	PEP Purpose and Guidelines	Warsaw, Hylton, Goodger, Coghlan
P	4	Deprecation of Standard Modules	Cannon, von Löwis
P	5	Guidelines for Language Evolution	Prescod
P	6	Bug Fix Releases	Aahz, Baxter
P	7	Style Guide for C Code	GvR, Warsaw
P	8	Style Guide for Python Code	GvR, Warsaw, Coghlan
P	10	Voting Guidelines	Warsaw
P	11	Removing support for little used platforms	von Löwis, Cannon
P	12	Sample reStructuredText PEP Template	Goodger, Warsaw, Cannon

https://www.python.org/dev/peps/

python

PEP 8
Style Guide for Python Code

- Code Lay-out
  - Indentation
  - Tabs or Spaces?
  - Maximum Line Length
  - Should a Line Break Before or After a Binary Operator?
  - Blank Lines
  - Source File Encoding
  - Imports
  - Module Level Dunder Names
- Naming Conventions
  - Overriding Principle
  - Descriptive: Naming Styles
  - Prescriptive: Naming Conventions
    - Names to Avoid
    - ASCII Compatibility
    - Package and Module Names
    - Class Names
    - Type Variable Names
    - Exception Names
    - Global Variable Names
    - Function and Variable Names
    - Function and Method Arguments
    - Method Names and Instance Variables
    - Constants
    - Designing for Inheritance
  - Public and Internal Interfaces

; que el
rcionad
acer co
nsitenci
nsistenc
ntro de
más importante.

https://www.python.org/dev/peps/pep-0008/

python

Frameworks para el desarrollo Web

- Pyramid
  - Curva rápida de aprendizaje
  - Desarrollo de proyectos para API’s RESTful
  - Desarrollo basado en prototips
  - Desarrollo de aplicaciones como CMS’s

https://trypyramid.com/

python

Frameworks para el desarrollo Web

- Botle
  - Desarrollos simples
  - Creación de un API web
  - Sin dependencias de bibliotecas adicionales

https://bottlepy.org/

Frameworks para el desarrollo Web

- Django
  - Desarrollo con múltiples herramientas integradas
  - Gran comunidad de desarrollo
  - Interfaz de administración
  - Lenguaje propio de plantillas
  - Documentación extensa

https://www.djangoproject.com/

Frameworks para el desarrollo Web

- Flask
  - Ideal para aprender a programar
  - Desarrollo enfocado en las buenas prácticas
  - Múltiples extensiones para ampliar la funcionalidad
  - Se puede combinar con diferentes bibliotecas para el manejo de bases de datos
  - Desarrollo de prototipos de forma y rápida

https://www.djangoproject.com/

Gracias

L.I. Arturo Rendón Cruz arendon@dgb.unam.mx