

A black and white photograph of a statue of a religious figure, likely a saint or pope, wearing a long robe and holding a book. The statue is positioned on the left side of the slide, partially obscured by a brown rectangular overlay.

INTRODUCCIÓN A LA PROGRAMACIÓN EN PYTHON

■ Miguel Orjuela



Universidad del
Rosario

Educación
Continua

Personas con
propósito
que ayudan a
transformar vidas



Universidad del
Rosario

Educación
Continua

Introducción a la programación en Python

2019

Sesión # 1

Variables, expresiones, operaciones básicas



Miguel Angel Orjuela Rocha
Ingeniero de Sistemas y Computación

Contenido

Visión general de conceptos esenciales de la programación en Python y las mecánicas del lenguaje (Tornillos y tuercas)

- Semántica
- Tipos básicos de datos

Contenido

- Semántica
 - Indentación
 - Objetos
 - Comentarios
 - Llamado de funciones y métodos
 - Pasar variables y argumentos
 - Referencias dinámicas, tipado
 - Atributos y métodos
 - Importar módulos
 - Operadores binarios y comparaciones
 - Objetos mutables e inmutables

Contenido

- Tipos básicos de datos
 - Numéricos
 - Strings
 - Bytes y Unicode
 - Booleanos
 - Type casting
 - None
 - Dates y times

Semántica

Semantica



Principios del lenguaje (pseudocódigo ejecutable):

1. Legibilidad
2. Sencillez
3. Claridad

Semántica

Python

```
stuff = ["Hello", "Hi", "6"]  
for i in stuff:  
    print(i)
```

Java

```
public class Test {  
    public static void main(String args[]) {  
        String array[] = {"Hello", "Hi", "6"};  
        for (String i : array) {  
            System.out.println(i);  
        }  
    }  
}
```

Indentación

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)
```

- Uso de espacios en blanco (o tabulaciones) para estructurar el código. No corchetes { }
- Dos puntos (:) indican el inicio de un bloque indentado
- Se recomienda usar cuatro espacios de indentación
- Las instrucciones de código no necesariamente tienen que terminar en punto y coma (;), pero su uso permite hacer múltiples instrucciones en la misma línea

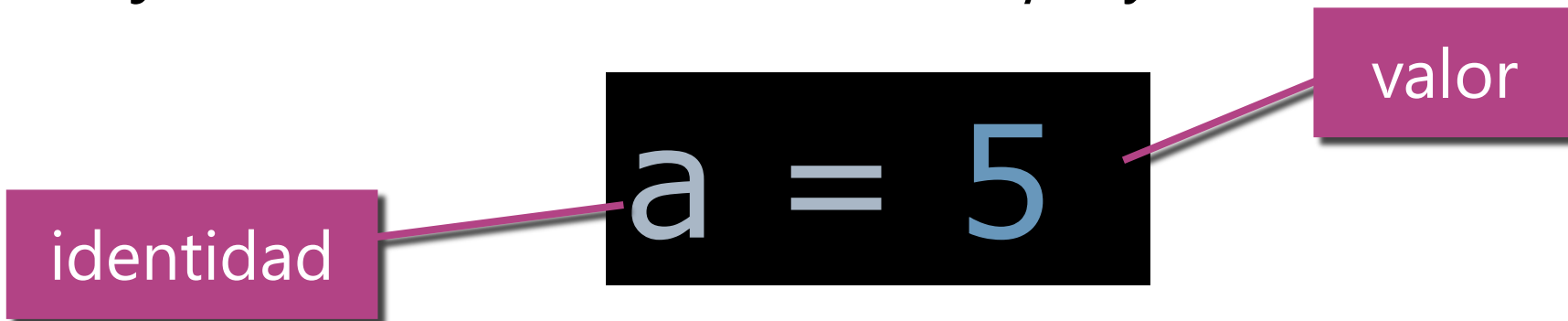
```
a = 5; b = 6; c = 7
```

Objetos

Los objetos son la abstracción de Python para los datos. Todos los datos en un programa de Python están representados por objetos o por relaciones entre objetos.

- Todo en Python es un objeto

Cada objeto tiene una *identidad*, un *tipo* y un *valor* asociado



Comentarios

Las instrucciones precedidas por `#` son ignoradas por el intérprete de Python

Usos de los comentarios:

- Agregar comentarios al código
- Excluir bloques de código sin borrarlos

```
a = 5;  
# b = 6;  
c = 7;  
print(a) # imprime 5
```

*Llamado de funciones y métodos

```
result = f(x,y,z)  
g()
```

```
obj.algun_metodo(x,y,z)
```

```
result = f(a,b,c,d=5,e='par')
```

Se llaman con paréntesis

Pueden tener cero o más argumentos

La mayoría de objetos tienen funciones anexas, conocidas como *métodos*

Los argumentos pueden ser por posición o por palabra clave

Pasar variables y argumentos

¿Qué valor toma **b**?

```
a = [1, 2, 3]
b = a

# comprobación
a.append(4)
b
```

En otros lenguajes, la asignación **b = a** haría que se copiaran los datos

En Python, **a** y **b** se refieren al mismo objeto



Binding: Unir nombre como un objeto

Pasar variables y argumentos

Cuando se pasan objetos como argumentos en una función, se crean variables locales nuevas referenciando los objetos originales

```
x = 10

def sumar_dos(numero):
    numero = numero + 2

sumar_dos(x)
```

Si se hace una asignación dentro de la función, el cambio no se ve fuera de ella

```
def agregar(lista, elemento):
    lista.append(elemento)

data = [1, 2, 3]
agregar(data, 4)

# comprobación
data
```

... sin embargo, es posible modificar un argumento *mutable*

Referencias dinámicas, tipado

Las *referencias* a objetos de Python no tienen un tipo de dato asociado

La información del tipo del objeto se guarda en el objeto

Java

```
int numberOfDays = 7;
```

Python

```
number_of_days = 7  
type(number_of_days)
```

Todo objeto en Python tiene un tipo específico.

Solo hay conversions implícitas en circunstancias obvias

```
a = 5  
b = '5'  
print(a + b)
```

```
a = 4.5  
b = 2  
print(a / b)
```


Referencias dinámicas, tipado

Para conocer el tipo de un objeto podemos usar los comandos

- `type`
- `isinstance`

```
a = '5'  
type(a)  
isinstance(a,int)
```

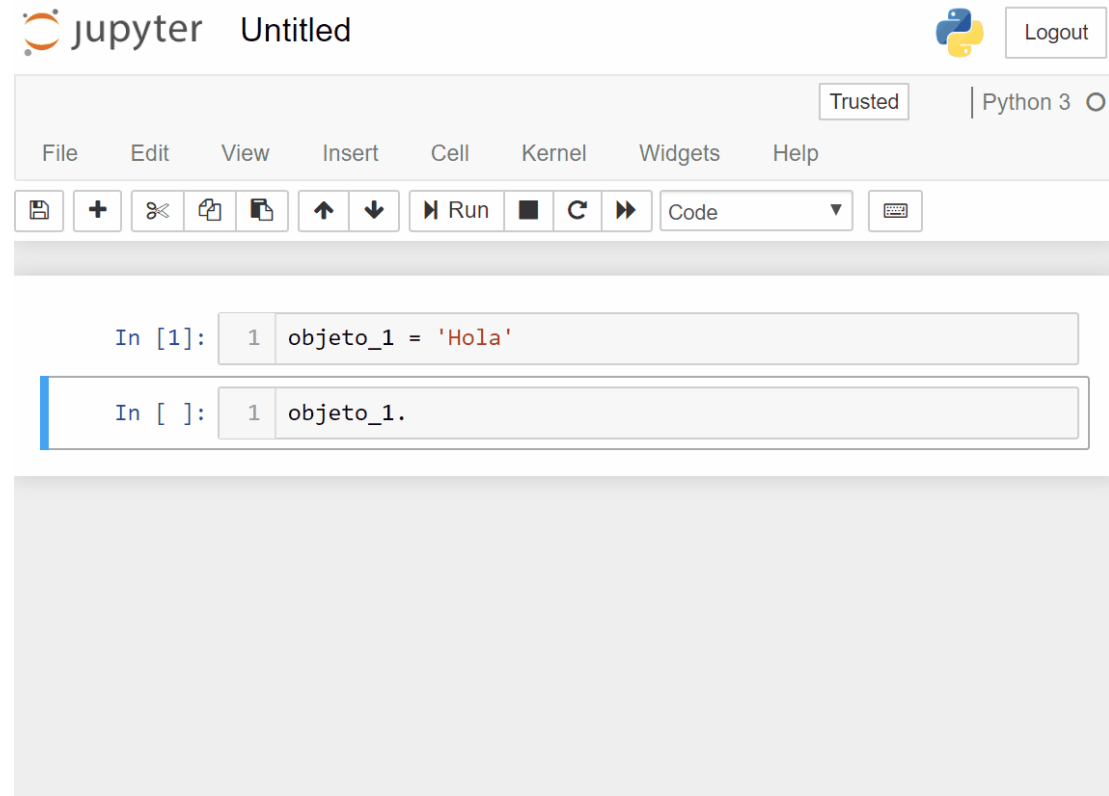
Atributos y métodos

Atributos: Objetos dentro de los objetos

Métodos: Funciones asociadas al objeto que acceden a los datos internos del objeto

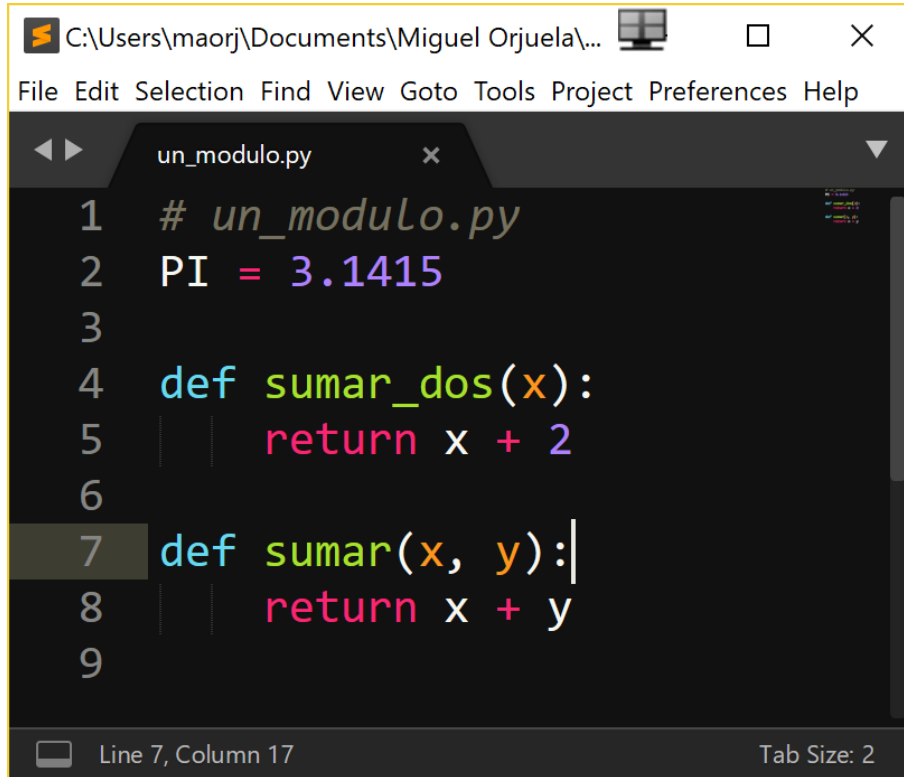
Se acceden con

```
objeto.nombre_atributo
```



Importar módulos

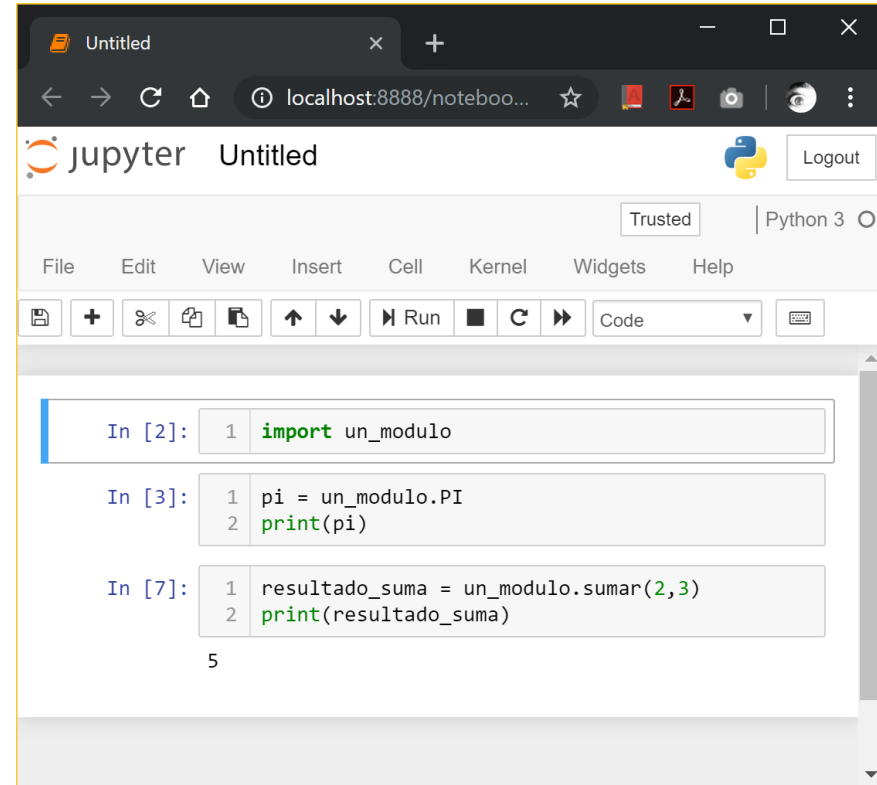
Un módulo es un archivo `.py` que contiene código Python



A screenshot of a text editor window titled 'un_modulo.py'. The code defines a module with a constant PI and two functions: sumar_dos and sumar. The code is as follows:

```
1 # un_modulo.py
2 PI = 3.1415
3
4 def sumar_dos(x):
5     return x + 2
6
7 def sumar(x, y):
8     return x + y
9
```

The status bar at the bottom indicates 'Line 7, Column 17' and 'Tab Size: 2'.



A screenshot of a Jupyter Notebook interface. The notebook contains three code cells demonstrating the use of the un_modulo module. The code is as follows:

```
In [2]: 1 import un_modulo

In [3]: 1 pi = un_modulo.PI
        2 print(pi)

In [7]: 1 resultado_suma = un_modulo.sumar(2,3)
        2 print(resultado_suma)

5
```

The output of the third cell is the number 5.

Importar módulos

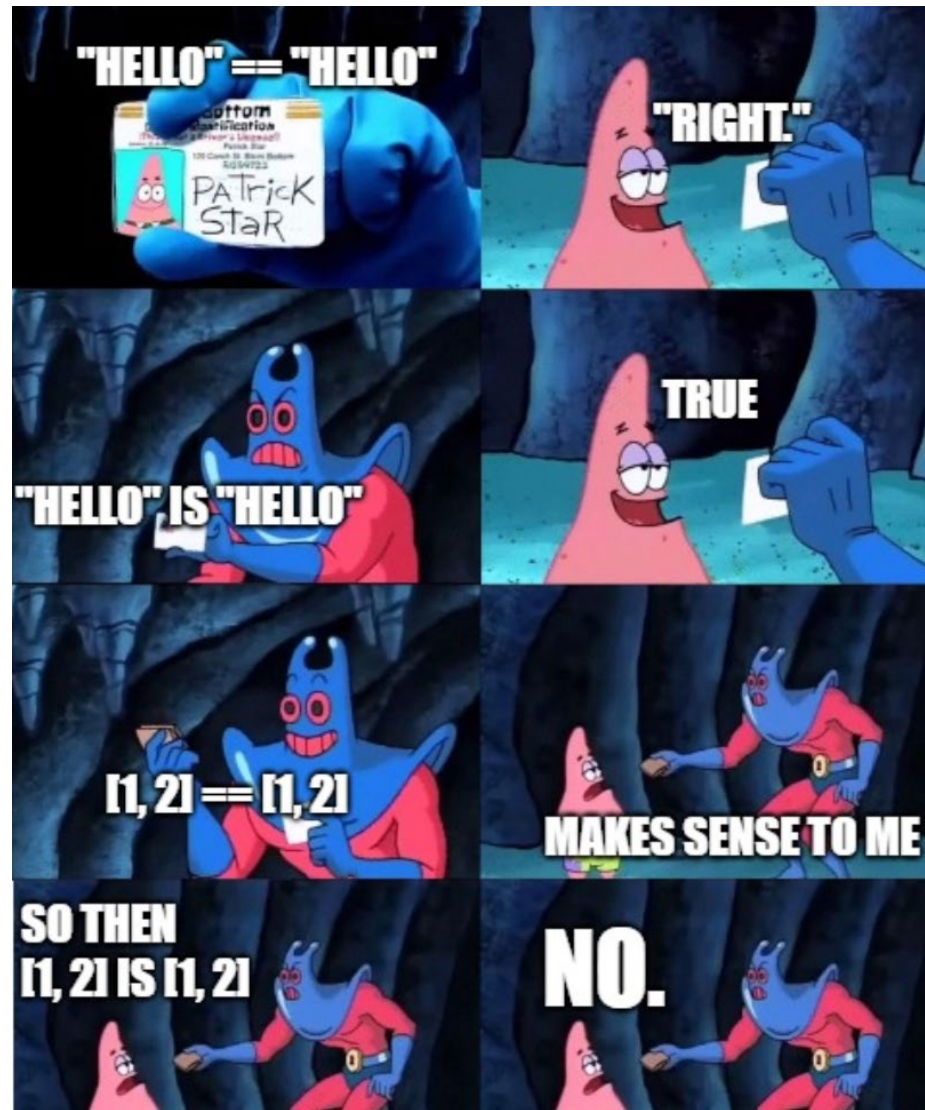
Existen varias formas de usar un módulo

```
import un_modulo
pi = un_modulo.PI
resultado_suma = un_modulo.sumar(2,3)
```

```
from un_modulo import sumar, sumar_dos, PI
pi_mas_dos = sumar_dos(PI)
```

```
import un_modulo as um
from un_modulo import sumar as suma_numeros, PI as cte_pi
res_1 = um.sumar_dos(cte_pi)
res_2 = suma_numeros(4,3)
```

Operadores binarios y comparaciones



Operadores binarios y comparaciones

Python es una calculadora

```
5 - 7  
12 + 21.5  
5 <= 2
```

Validar si dos referencias apuntan al mismo objeto con **is**

```
a = [1,2,3]  
b = a  
a is b  
a is not b
```

a →

b →

list

[1, 2, 3]

Operadores binarios y comparaciones

Tip: El comando `list()` crea una nueva lista



Operadores binarios y comparaciones

El objeto **None** es el valor nulo de Python

En Python solo existe una instancia del objeto **None**



Operadores binarios y comparaciones

Aritméticos

OPERATOR	DESCRIPTION	SYNTAX
+	Addition: adds two operands	x + y
-	Subtraction: subtracts two operands	x - y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when first operand is divided by the second	x % y

Relacionales

OPERATOR	DESCRIPTION	SYNTAX
>	Greater than: True if left operand is greater than the right	x > y
<	Less than: True if left operand is less than the right	x < y
==	Equal to: True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to: True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to: True if left operand is less than or equal to the right	x <= y

Lógicos

OPERATOR	DESCRIPTION	SYNTAX
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if operand is false	not x

De asignación

OPERATOR	DESCRIPTION	SYNTAX
=	Assign value of right side of expression to left side operand	x = y + z
+=	Add AND: Add right side operand with left side operand and then assign to left operand	a+=b a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b a=a-b
=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a=b a=a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b a=a/b
%=	Modulus AND: Takes modulus using left and right operands and assign result to left operand	a%=b a=a%b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b a=a//b
=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a=b a=a**b

<https://www.geeksforgeeks.org/basic-operators-python/>

Objetos mutables e inmutables

Mutable: El objeto o valores que el objeto contienen pueden ser modificados, p.ej. *listas*, *diccionarios*, *arrays* de NumPy, y tipos definidos por el usuario

```
una_lista = ['hola', 2, [4, 5]]  
print(una_lista)  
una_lista[2] = (3, 4)  
print(una_lista)
```

Otros, como *strings* y *tuplas* no pueden ser modificados

```
una_tupla = (3, 5, (4, 5))  
print(una_tupla)  
una_tupla[1] = 'adios'  
print(una_tupla)
```

`una_lista[i]`



Pedir objeto en la posición *i*

Tipos básicos de datos

Tipos básicos de datos

Python tiene un conjunto pequeño de tipos de datos en su librería estándar para manejar datos numéricos, strings (cadenas de caracteres), booleanos (cierto, falso), fechas y horas.

Estos tipos de datos se conocen como *escalares* (en contraposición a compuestos)

- None: valor nulo
- str: cadena de caracteres Unicode(codificación UTF-8)
- bytes: ASCII bytes (o Unicode codificados como bytes)
- float: Números punto flotante de doble precisión (64 bits)
- bool: Toman valores True o False
- int: Entero con signo

Numéricos

int

```
valor_entero = 12345  
valor_entero ** 2
```

float

```
valor_float = 7.2355  
valor_float_2 = 2.43e-5
```

Strings

Son cadenas de caracteres, se pueden escribir con comillas sencillas o dobles

```
a = 'una forma de hacer strings'  
b = "otra forma de hacer strings"
```

```
c = """  
este es un string  
multi  
linea  
"""
```

```
a.count(" ")  
a.count("\n")
```

Strings

Los strings son inmutables

```
x = "Los strings son inmutables"  
x[4]  
x[4] = 'S'
```

Pero tienen métodos que permiten su modificación

```
x = x.replace('Los strings', 'Las cadenas de caracteres')
```

También podemos convertir muchos objetos a string

```
num_float = 7.4234234  
num_string = str(num_float)
```

Strings

Los strings pueden ser tratados como secuencias de caracteres

```
s = 'python'  
s[:3]  
list(s)
```

s[:3]



slicing (sacar tajadas)

Strings

Los caracteres especiales se escriben con backslash (\)

```
a.count("\n")
```

Si se quiere hacer un string con algún caracter especial, hay que escaparlo, anteponiendo un backslash

```
a = "Para \"saltar\" de línea oprimimos \\n"
```

Para evitar escapar los caracteres, se puede anteponer una **r** al string

```
s = r'este\string\no\tiene\caracteres\especiales'
```



raw

String

Concatenar: Unir dos strings

```
a = "Los strings se"  
b = "pueden concatenar"  
a + b
```

String

Template: Plantilla de string en la que indicamos el formato de algunas partes.

```
template = '{0:0.2f} {1:s} equivalen a {2:d} USD'
```

Punto flotante

String

Entero

Las partes formateadas se pasan como argumentos en el método format

```
template.format(3450,'COP',1)
```

Bytes y Unicode

Codificación de caracteres: Método que permite convertir un carácter de un lenguaje natural (como el de un alfabeto o silabario) en un símbolo de otro sistema de representación, como un número o una secuencia de pulsos eléctricos en un sistema electrónico, aplicando normas o reglas de codificación.

Definen la forma en la que se codifica un carácter dado en un símbolo en otro sistema de representación.

- Morse
- Norma ASCII
- UTF-8
- Unicode

https://es.wikipedia.org/wiki/Codificaci%C3%B3n_de_caracteres

Bytes y Unicode

Acentos y tildes					
carácter	ISO-8859-1	UTF-8		UTF-16	
á	0xe1	0xc3	0xa1	0x00	0xe1
Á	0xc1	0xc3	0x81	0x00	0xc1
é	0xe9	0xc3	0xa9	0x00	0xe9
É	0xc9	0xc3	0x89	0x00	0xc9
í	0xed	0xc3	0xad	0x00	0xed
Í	0xcd	0xc3	0x8d	0x00	0xcd
ó	0xf3	0xc3	0xb3	0x00	0xf3
Ó	0xd3	0xc3	0x93	0x00	0xd3
ú	0xfa	0xc3	0xba	0x00	0xfa
Ú	0xda	0xc3	0x9a	0x00	0xda
ü	0xfc	0xc3	0xbc	0x00	0xfc
Ü	0xdc	0xc3	0x9c	0x00	0xdc
ñ	0xf1	0xc3	0xb1	0x00	0xf1
Ñ	0xd1	0xc3	0x91	0x00	0xd1

```
palabra = 'español'
```

```
palabra_utf8 = palabra.encode('utf-8')  
type(palabra_utf8)  
palabra_utf8.decode('utf-8')
```

```
palabra_latin1 = palabra.encode('latin1')  
type(palabra_latin1)  
palabra_latin1.decode('latin1')
```

[https://es.wikipedia.org/wiki/Codificaci%C3%B3n de caracteres](https://es.wikipedia.org/wiki/Codificaci%C3%B3n_de_caracteres)

Bytes y Unicode

Es más común encontrar objetos de bytes en el contexto de trabajar con archivos, donde es posible que no se desee decodificar implícitamente todos los datos en cadenas Unicode. Aunque es posible que rara vez tenga que hacerlo, puede definir sus propios literales de bytes prefijando un string con **b**

bytes

```
bytes_val = b'Estos son bytes'  
type(bytes_val)  
val_decodificado = bytes_val.decode('utf-8')  
type(val_decodificado)
```

Booleanos

True y False

Las comparaciones y otras expresiones condicionales evalúan a **True** o a **False**

```
5 == 3 + 2
True and False
True or False
```

OPERATOR	DESCRIPTION	SYNTAX
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if operand is false	not x

Type casting

Casting: Convertir valores de un tipo a otro

Los tipos `str`, `bool`, `int` y `float` son también funciones que convierten valores de un tipo a otro

```
s = '3.14159'  
s_float = float(s)  
type(s_float)  
s_int(s_float)  
type(s_int)  
bool(s_float)  
bool(0)
```


None

Valor **nulo** de Python

Si una función no retorna explícitamente nada, entonces retorna **None**

```
a = None
a is None
b = 3
b is not None
```

None es también usado como argumentos por defecto en funciones

```
def suma_talvez_multiplica(a, b, c = None):
    resultado = a + b
    if c is not None:
        resultado *= c
    return resultado
suma_talvez_multiplica(2,3)
suma_talvez_multiplica(2,3,2)
```

Dates y times

```
from datetime import datetime, date, time
```


El modulo datetime provee las clases datetime, date y time

```
dt = datetime(2019,06,12,17,30,00)  
dt.day  
dt.minute  
dt.date()  
dt.time()
```

Dates y times

Convertir de datetime a string

```
dt.strftime('%m/%d/%Y %H:%M')
```



formato

Convertir de string a datetime

```
datetime.strptime('20091031', '%Y%m%d')
```



formato

Dates y times

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018

https://www.w3schools.com/python/python_datetime.asp

Dates y times

Si se restan dos `datetimes`, el resultado es un `timedelta`

```
dt = datetime(2018,6,12,17,30,00)
dt2 = datetime(2019,6,14,17,30,20)
delta = dt2 - dt
```

La salida `timedelta(367,20)` indica una distancia de 367 días y 20 segundos

A un `datetime` se le puede sumar un `timedelta`

```
dt2 + delta
```

Resumen

- La sintaxis de Python es compacta respecto a otros lenguajes
- Python cuenta con clases y funciones dentro de ellas para manejar tipos básicos de datos
- Convertir objetos de un tipo a otro es posible

A continuación

Control de flujo