

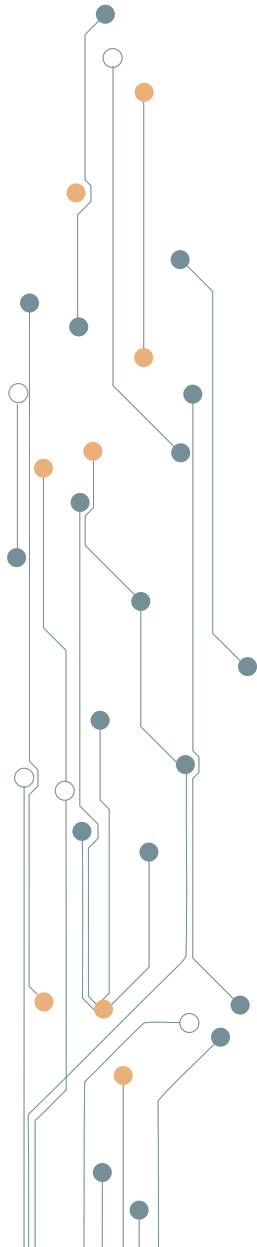


# Introducción a la programación con Phyton

*Telefónica*

**EDUCACIÓN DIGITAL**

# Índice



1   Introducción	3
2   Instalación y configuración	4
3   Introducción al lenguaje de programación	7
3.1   Tipos de datos en Python	7
3.2   Operadores aritméticos	11
3.3   Operadores relacionales (Comparación)	12
3.4   Operadores Lógicos (Condicionales)	13
3.5   Operadores de Asignación	14
3.6   Sentencia IF	15
3.7   Bucles While	16
3.8   Bucles For	17
3.9   Funciones	18
3.10   Manejo de Excepciones	19
3.11   Operaciones con estructuras de datos y cadenas	20

# 1. Introducción

- Este módulo tiene como objetivo el disponer al alumno de los conocimientos básicos y de las herramientas necesarias para realizar diversas tareas de Seguridad que se pueden automatizar mediante scripts escritos en Python.
- Este módulo no tiene como objetivo ser un manual de aprendizaje del lenguaje de programación Python. Si no que se da por supuesto que el alumno posee conocimientos de programación previos (en Python u otros lenguajes) y que se enfrenta a este módulo para aprender diferentes técnicas de Seguridad que poder realizar con dicho lenguaje.



## 2. Instalación y configuración

Python es un lenguaje de programación interpretado el cual es multiparadigma ya que permite la programación imperativa, la programación orientada a objetos y la programación funcional.

Un programa en Python no se compila para obtener un ejecutable en código máquina, si no que es un script en texto plano que necesita de un intérprete para poder ejecutarse.

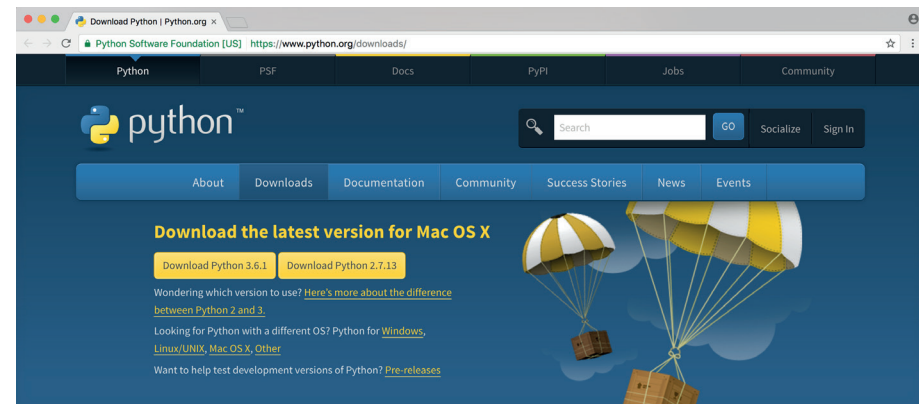
Este intérprete es multiplataforma y está disponible en los principales sistemas operativos (Windows, Unix, MacOS, etc). Cualquier script de Python se puede ejecutar en cualquier plataforma que contenga el intérprete en la misma versión.

Para este modulo se va a usar la versión de Python 3.5.X, en vez de la 2.7 ya que es la última versión desarrollada y es la que más va a perdurar más en el tiempo.

La mayoría de librerías utilizadas en el módulo ya cuentan con soporte para la versión 3.5.X. En los apartados que la librería no esté soportada se indicará y se mostrará una alternativa a la misma.

La mejor forma de instalación de Python es acceder a la sección descargas de su página oficial <https://www.python.org/downloads/> y descargarse la última versión estable para el sistema operativo destino.

Para los sistemas operativos Windows y MacOS simplemente habrá que ejecutar el fichero de instalación.



Looking for a specific release?  
Python releases by version number:

Release version	Release date		Click for more
Python 3.6.1	2017-03-21	<a href="#">Download</a>	<a href="#">Release Notes</a>
Python 3.4.6	2017-01-17	<a href="#">Download</a>	<a href="#">Release Notes</a>
Python 3.5.3	2017-01-17	<a href="#">Download</a>	<a href="#">Release Notes</a>
Python 3.4.6	2016-12-23	<a href="#">Download</a>	<a href="#">Release Notes</a>
Python 2.7.13	2016-12-17	<a href="#">Download</a>	<a href="#">Release Notes</a>
Python 3.4.5	2016-06-27	<a href="#">Download</a>	<a href="#">Release Notes</a>
Python 3.5.2	2016-06-27	<a href="#">Download</a>	<a href="#">Release Notes</a>

[View older releases](#)

Para los sistemas Linux habrá que descomprimir el archivo comprimido descargado y ejecutar los siguientes comandos:

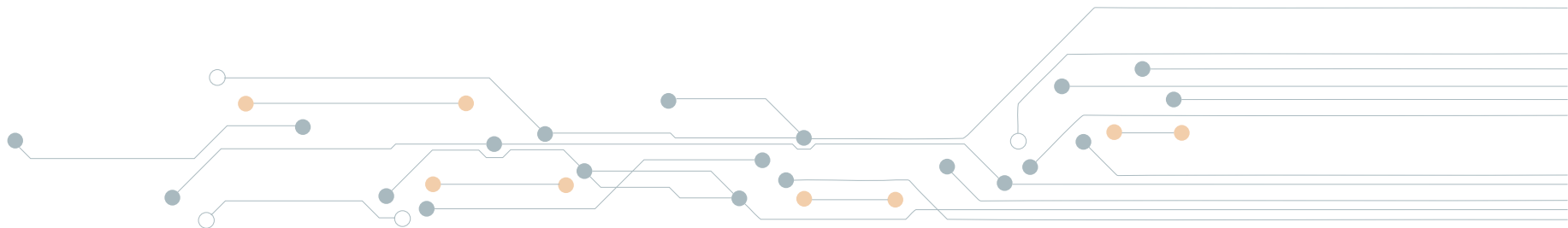
```
sudo tar xzf Python-3.X.X.tgz  
cd Python-3.X.X  
sudo ./configure  
sudo make altinstall
```

Además de descargar Python, vamos a descargar una herramienta que gestiona la descarga e instalación de librerías de una forma muy simple y rápida. Esta herramienta es PIP (Python Install Package), es un script escrito en Python y su funcionamiento es consultar en el repositorio de librerías PyPI (Python Package Index) la mejor versión de la librería en base a la versión de Python instalada en el equipo.

En algunos paquetes de instalación de Python se descarga de forma conjunta esta librería, pero en el resto de los casos habrá que descargarla aparte. El método de instalación que es común para todos los sistemas operativos es descargarse el script `get-pip.py` de su Github oficial <https://raw.githubusercontent.com/pypa/pip/master/contrib/get-pip.py>

Una vez en el equipo, simplemente es necesario ejecutar el script con Python:

```
python get-pip.py
```



Para comprobar si se ha instalado correctamente habrá que ejecutar en consola el comando:

```
pip
```

Si ya se disponía de una versión de PIP instalada, es muy recomendable actualizarla antes de empezar este módulo. Para ello solo será necesario ejecutar en consola el siguiente comando:

```
python -m pip install --upgrade pip
```

Una vez actualizado PIP ya está configurado el entorno con todo lo necesario para realizar los ejemplos de este módulo.



## 3. Introducción al lenguaje de programación

Como se comentaba anteriormente, este módulo no tiene como objetivo ser un manual de aprendizaje del lenguaje de programación Python. Si no que se da por supuesto que el alumno posee conocimientos de programación previos (en Python u otros lenguajes) y que se enfrenta a este módulo para aprender diferentes técnicas de Seguridad que poder realizar con dicho lenguaje.

Por este motivo, este apartado "Introducción al lenguaje de programación" tiene como objetivo de servir como guía de referencia para mostrar de una forma clara y simple cómo se realizan las diferentes acciones en este lenguaje de programación y las pequeñas particularidades que tiene.

### 3.1 | Tipos de datos en Python

En el lenguaje de programación Python no se especifica el tipo de una variable al crearla, sino que la infiere del valor que se le asigna:

```
c = "Hola Mundo"  
e = 23
```

Para conocer el tipo de una variable determinada en tiempo de ejecución, se utiliza el método `type()` del propio lenguaje de programación:

```
type(c)  
<class 'str'>  
  
type(e)  
<class 'int'>
```

### Tipos numéricos:

- En Python hay varios tipos numéricos:
- Enteros: Los números enteros son aquellos que no tienen decimales, tanto positivos como negativos. Se pueden representar mediante el tipo int (de integer, entero) o el tipo long (largo)
- Reales: Los números reales son los que tienen decimales. Se representan mediante el tipo float.
- Complejos: Los números complejos son aquellos que tienen parte imaginaria. Se representan mediante el tipo complex.

```
entero = 23
entero = 23L # Tipo long
entero = 027 # Base octal
entero = 0x17 # Base hexadecimal

real = 0.2703
real = 0.1e-3

complejo = 2.1 + 7.8j
```

### Tipo cadena:

Las cadenas son simplemente texto encerrado entre comillas simples ('cadena') o dobles ("cadena"). Dentro de las comillas se pueden añadir caracteres especiales escapándolos con '\', como '\n' o '\t'.

Además, se puede representar las cadenas en varias codificaciones como por ejemplo Unicode precediendo la cadena de una 'u' o en formato Raw precediéndola de 'r'. El formato Raw no sustituye las barras invertidas (\) al ser utilizadas, por lo que es útil para usarlas en las expresiones regulares.

```
cadena = "Texto entre \n\t comillas simples"
cadena = " " "cadena
multilínea
...
línea N
" " "

unicode = u"äóè"
raw = r"\n"
```



### Tipo booleano:

El tipo booleano solo puede tomar dos valores: True (cierto) y False (falso). Se puede representar mediante el tipo bool.

```
aT = True  
aF = False
```

### Tipo conjunto:

Un conjunto, es una colección no ordenada y sin elementos repetidos. Los usos básicos de éstos incluyen verificación de pertenencia y eliminación de entradas duplicadas.

```
conjunto = set(['madrid', 'barcelona', 'sevilla', 'valencia',  
               'toledo', 'avila'])  
existe_madrid = 'madrid' in conjunto
```

### Tipo lista:

La lista en Python son variables que almacenan un listado de elementos, internamente cada posición puede ser un tipo de datos distinto.

```
lista = ['madrid', "tres", 4, True, ["uno", 10]]  
primer_elemento = lista[0]  
lista[4] = False  
sublista = lista[0:3] # Obtener un rango de la lista
```

### Tipo tupla:

Una tupla es una lista inmutable. Una tupla no puede modificarse de ningún modo después de su creación.

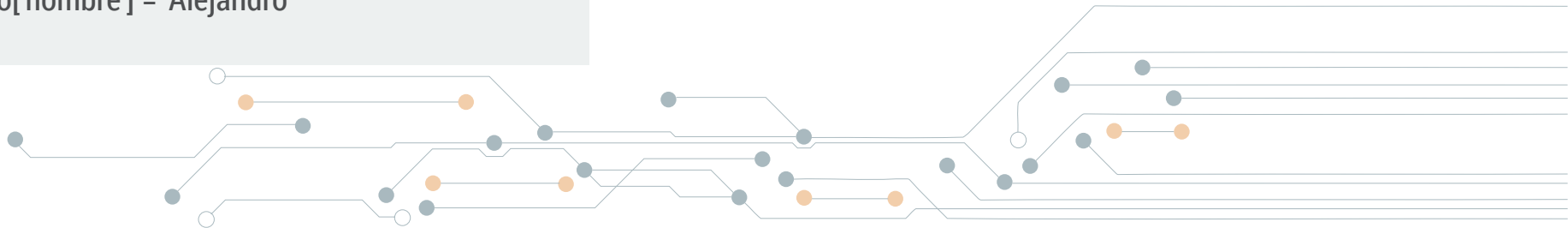
```
tupla = 'madrid', "tres", 4  
x, y, z = tupla  
otra_tupla = tupla, (1, 2, 3, 4, 5)  
primer_elemento = tupla[0]
```



### Tipo diccionario:

Un diccionario es una estructura con una colección de pares clave/valor los cuales tienen una relación uno a uno.

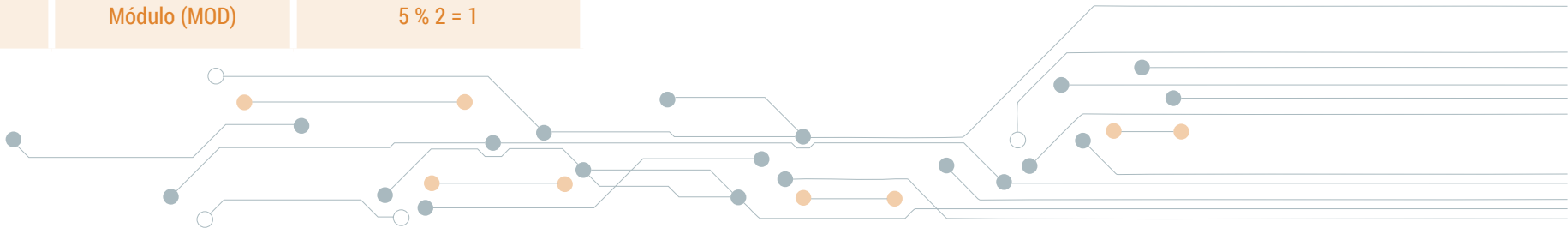
```
diccionario = {  
    "nombres": "Antonio",  
    "apellidos": "Del Pozo",  
    "fecha_nacimiento": "03121980",  
    "lugar_nacimiento": "Madrid",  
    "nacionalidad": "Española",  
    "estado_civil": "Soltero"  
}  
print ("Claves:", diccionario.keys())  
print ("Valores:", diccionario.values())  
print ("Elementos:", diccionario.items())  
diccionario['nombre']  
diccionario['nombre'] = 'Alejandro'
```



## 3.2 | Operadores aritméticos

Los operadores aritméticos son aquellos que nos permiten realizar cálculos con tipos numéricos. Los operadores aritméticos en Python son:

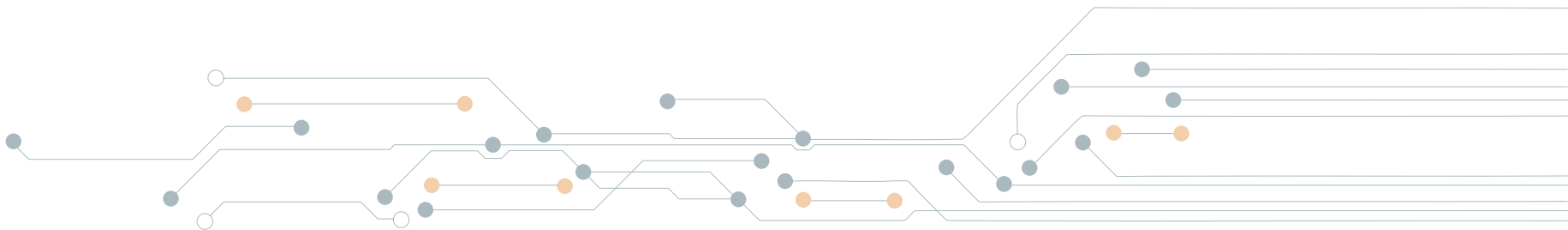
Operador	Descripción	Ejemplo
+	Suma	$2 + 2 = 4$
-	Resta	$2 - 1 = 1$
-	Negación	$-6 = -6$
*	Multiplicación	$2 * 3 = 6$
**	Exponente	$2 ** 3 = 8$
/	División	$5.0 / 2 = 2.5$
//	División entera (DIV)	$5 // 2 = 2$
%	Módulo (MOD)	$5 \% 2 = 1$



## 3.3 | Operadores relacionales (Comparación)

Los operadores relacionales nos permiten realizar acciones con diferentes tipos de datos dando siempre como resultado un tipo bool:

Operador	Descripción	Ejemplo
<code>==</code>	Igual	<code>2 == 2 #True</code>
<code>!=</code>	Distinto	<code>2 != 3 #True</code>
<code>&lt;</code>	Menor que	<code>4 &lt; 3 #False</code>
<code>&gt;</code>	Mayor que	<code>4 &gt; 3 #True</code>
<code>&lt;=</code>	Menor o igual	<code>4 &lt;= 4 #True</code>
<code>&gt;=</code>	Mayor o igual	<code>3 &gt;= 4 #False</code>



## 3.4 | Operadores Lógicos (Condicionales)

Los operadores lógicos nos permiten trabajar con tipos booleanos:

Operador	Descripción	Ejemplo
and	A y B	True and False #False
or	A o B	True or False #True
not	No A	Not True #False

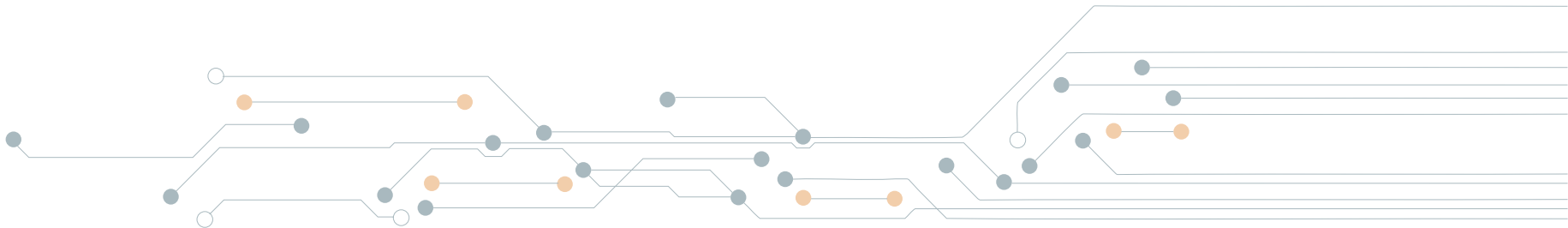


## 3.5 | Operadores de Asignación

Los operadores de asignación se usan para asignar un valor a una variable. El operador de asignación básico es '=' el cual almacena el valor o variable del lado derecho a la variable del lado izquierdo.

Aparte, existen otros tipos de operadores de asignación los cuales realizan una operación aritmética con la variable a asignar antes de almacenar el dato. Los diferentes operadores de asignación son los siguientes:

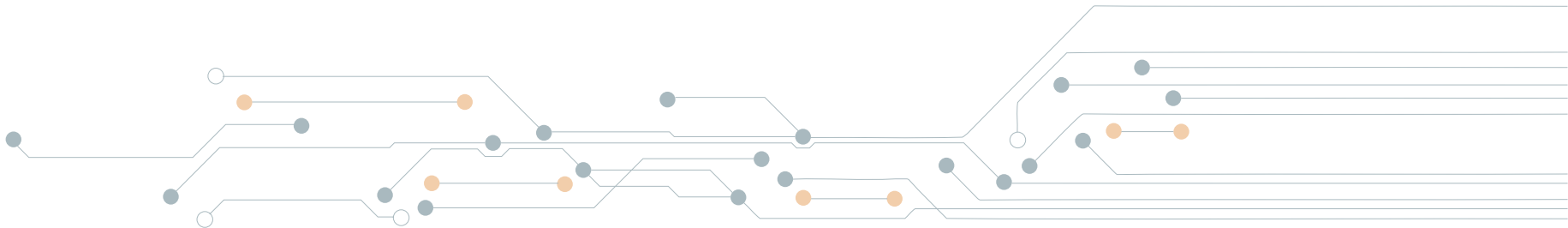
- +=
- -=
- \*=
- /=
- \*\*=
- //=
- %=



## 3.6 | Sentencia IF

Esta sentencia se usa para tomar decisiones en base a cualquier operación lógica cuyo resultado sea un booleano, permitiendo derivar en 2 flujos funcionales dentro del mismo programa. El formato de una sentencia IF en Python es el siguiente:

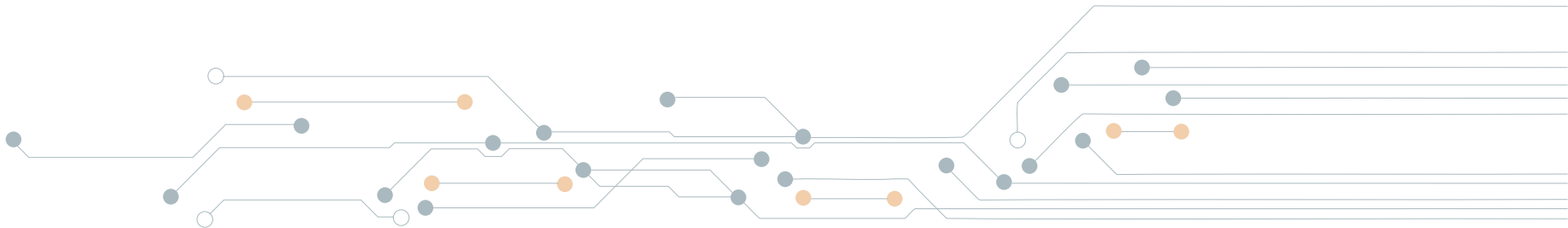
```
if numero < 0:  
    print('Numero negativo')  
elif numero == 0:  
    print('Numero es Cero')  
elif numero == 1:  
    print('Numero Simple')  
else:  
    print('Mayor que uno')
```



## 3.7 | Bucles While

Esta sentencia se usa para crear bucles en el código en base al resultado de una expresión lógica que determina si finaliza el bucle:

```
suma = 0
numero = 1
while numero <= 10:
    suma = numero + suma
    numero = numero + 1
```





## 3.8 | Bucles For

Esta sentencia se usa para crear bucles en el código pero a diferencia de los bucles While, esta se ejecuta un número determinado de veces. A diferencia de otros lenguajes de programación, los bucles For se ejecutan sobre una secuencia de elementos (listas o tuplas):

```
animales = ['perro', 'gato', 'leon', 'lobo']
```

```
for animal in animales:
```

```
    print (animal)
```

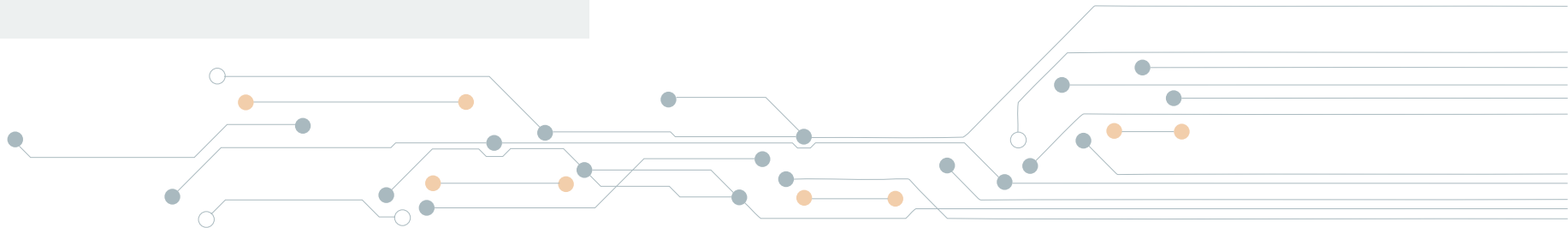
```
conexion_bd = "127.0.0.1", "root", "123456", "nomina"
```

```
for parametro in conexion_bd:
```

```
    print parametro
```

```
for numero in range(10):
```

```
    print (numero)
```



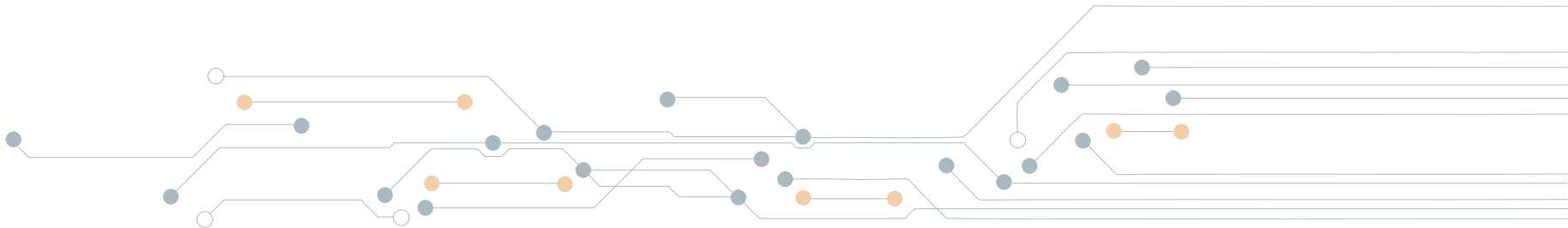
## 3.9 | Funciones

Las funciones son útiles para reutilizar un mismo código en diferentes partes de un Script. Se define una función con unos parámetros de entrada, se ejecuta un código determinado en base a esos parámetros y se devuelve un resultado como salida:

```
def suma(numero1,numero2):  
    print numero1 + numero2
```

Para invocar a estas funciones:

```
suma = suma(10, 5)  
print(suma)
```



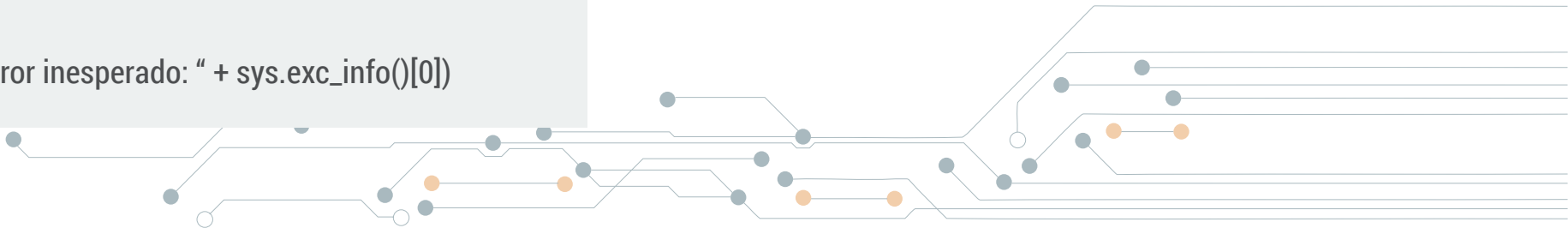
## 3.10 | Manejo de Excepciones

Los errores surgidos en tiempo de ejecución que rompen el flujo habitual de un programa se denominan Excepciones. Estas excepciones se pueden manejar y tratar para que el programa no finalice de forma inesperada. Cada excepción capturada es de una clase concreta, por lo que se puede hacer un tratamiento a bajo nivel de las mismas:

```
import sys
try:
    f = open('archivo.txt')
    s = f.readline()
    i = int(s.strip())
except IOError as (errno, strerror):
    print ("Error E/S " + errno + " - Error:" + strerror)
except ValueError:
    print ("No pude convertir el dato a un entero.")
except:
    print ("Error inesperado: " + sys.exc_info()[0])
```

Del mismo modo que se puede tratar las excepciones, también es posible elevarlas en un determinado punto del código:

```
raise NameError('Error')
```



## 3.11 | Operaciones con estructuras de datos y cadenas

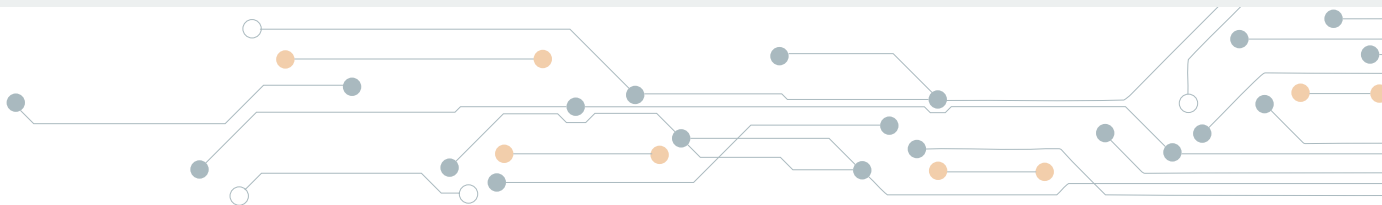
Es importante conocer todos los operadores disponibles para el manejo de tipos complejos y cadenas, ya que con esto facilita el desarrollo de los scripts. En los siguientes ejemplos se muestran las operaciones más útiles de cada uno de los tipos.

### Tipo lista:

```
lista = list() # Declaración de una lista
len(lista) # Número de elementos de la lista
lista.append(x) # Agrega un elemento al final de la lista.
lista.insert(pos, x) # Inserta un elemento en una posición determinada
lista.extend(lista2) # Une dos listas
lista.remove(x) # Borra el primer elemento de la lista cuyo valor sea x.
lista.pop(pos) # Borra el elemento en la posición dada de la lista, y lo devuelve.
lista.index(x) # Devuelve el índice en la lista del primer elemento cuyo valor sea x.
lista.count(x) # Devuelve el número de veces que x aparece en la lista.
lista.sort(cmp=None, key=None, reverse=False) # Ordena los ítems de la lista
lista.reverse() # Invierte los elementos de la lista.
listaCopia = lista.copy() # Devuelve una copia de la lista
```

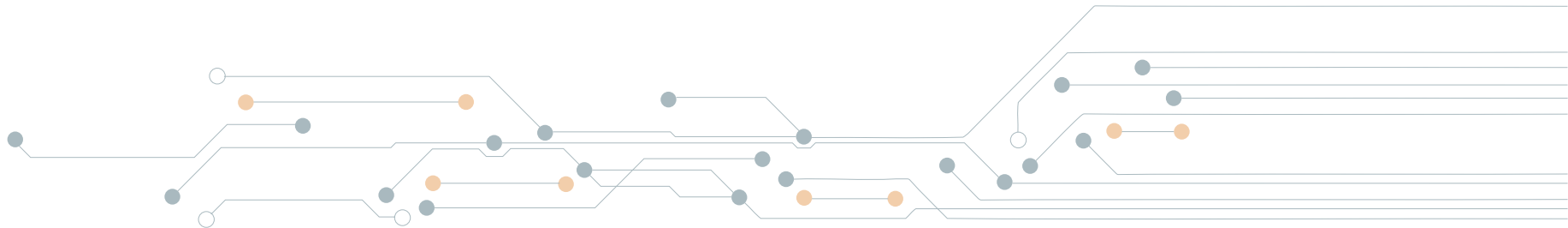
### Tipo diccionario:

```
diccionario = dict() # Declaración de un diccionario
len(dict) # Numero de elementos que tiene el diccionario
cmp (dict1,dict2) # Compara el número de elementos distintos que tienen los dos
dict.keys() # Devuelve una lista con las claves del diccionario
dict.values() # Devuelve una lista con los valores del diccionario
dict.get(key, default=None) # Devuelve el valor del elemento con clave key. Si no devuelve default
dict.setdefault(key, default=None) # Inserta un elemento en el diccionario clave:valor. Si la clave existe no lo inserta
dict['key'] = 'value' # Insertamos un elemento en el diccionario con su clave:valor
dict.pop('key',None) # Eliminamos el elemento del diccionario con clave key
dict.copy() # Devuelve la copia de un diccionario dict2 = dict.copy()
dict.clear() # Elimina todos los elementos de un diccionario
dict.fromkeys(list, defaultValue) # Crea un diccionario poniendo como claves las que hay en la lista y los valores por defecto si se les pasa
dict.has_key(key) # Devuelve true si existe la clave. Si no devuelve false
dict.items() # devuelve un lista de tuplas formadas por los pares clave:valor
dict.update(dict2) # Añade los elementos de un diccionario a otro
```



### Tipo cadena:

```
cadena = "Esto es una cadena"
len(cadena) # Devuelve la longitud de una cadena
cadena.find("t") # Devuelve el índice correspondiente al primer match con la cadena introducida
cadena.replace("una", "unas") # Reemplaza una serie de caracteres por otros
cadena.strip() # Elimina los espacios en blanco al inicio y final de la cadena
cadena.lstrip() # Elimina los espacios en blanco al inicio de la cadena
cadena.rstrip() # Elimina los espacios en blanco al final de la cadena
cadena.upper() # Convierte todos los caracteres a mayúsculas
cadena.lower() # Convierte todos los caracteres a minúsculas
cadena.capitalize() # Convierte el primer carácter a mayúsculas
cadena.split(" ") # Divide una cadena cortando por un determinado carácter. El resultado es una lista
caracter.join(nombres) # Junta una lista de elemento en una cadena separándolos por el carácter
```



*Telefónica* EDUCACIÓN DIGITAL