**class** Empleado(Persona):  
 *# Organizan/administran actividades.* **def** \_\_init\_\_(self, dni=**None**, nombre=**None**, apellidos=**None**, email=**None**, fecha\_nacimiento=**None**, departamento=**None**,  
 fecha\_contratacion=**None**, id=**None**):  
 Persona.\_\_init\_\_(self, dni, nombre, apellidos, email, fecha\_nacimiento)  
 **if** id **is None**:  
 self.id = Fun.generar\_id\_empleado()  
 **else**:  
 self.id = id  
 *#...*

**def** \_\_str\_\_(self):  
 **return "{0} - {1}, {2}."**.format(str(self.id), str(self.apellidos), str(self.nombre))

**Tuplas ->** son como las listas, pero los datos no pueden ser modificados

tupla = ("Hola", "Que tal", 5) -> creación de una tupla, se usan los paréntesis ()

**Listas ->** guardan datos de diferentes tipos y pueden ser modificados

lista = ["Hola", "Que tal", 5] -> creación de una lista, se usan los corchetes []

len(lista) -> longitud lista

lista[0] -> accede al objeto 0 de la lista

lista3 = lista1 + lista2 -> une dos listas

lista\_vacia = [] -> crea una lista vacía

lista3 = lista2 -> no hace una copia, sino que señala a los mismos elementos

for x in lista: -> para recorrer la lista

print(x)

if 2 in lista: -> para buscar en la lista

print("El 2 está en la lista")

lista.append(a) -> añade a al final de la lista

lista.insert(indice, a) -> añade a en el indice (mientras sea >= 0 y <= len(lista)), desplaza los demás a la derecha

lista.extend(lista2) -> añade la lista 2 al final de la lista, es igual que lista += lista2

lista.index(a) -> busca a y devuelve su índice, si no existe, lanza error "ValueError"

lista.remove(a) -> busca a

lista.sort() -> ordena la lista

lista.reverse() -> invierte el orden de la lista

lista.pop(indice) -> quita el elemento del indice y lo devuelve

lista.clear() -> deja vacía la lista

lista.count(a) -> devuelve el número de veces que existe a en la lista

print(lista[1:3]) -> mostrará los elementos 1 y 2 de la lista (el segundo y el tercero), funciona como el range

**Diccionarios ->** colección no ordenada de valores a los que se accede a través de una clave

Las claves son únicas, los valores pueden repetirse. No puedes acceder a una clave por el valor

diccionario = {"Valor1":1, "Valor2":2, "Valor3":3, 4:4} -> creación de un diccionario, se usan las llaves {}

diccionario = {} -> diccionario vacío

diccionario["Valor1"] = [1] -> podemos añadir elementos manualmente, asignamos clave y su valor o valores

diccionario["Valor2"] = [2] -> si la clave ya existe, en vez de añadirla al diccionario, sustituiría su valor

diccionario[3] = [3, 4, 5]

for x, valores in diccionario.items():

print(x, ":", valores)

dict() -> sirve también para crear un diccionario

diccionario = dict(nombre="Dani", apellido="Tamargo", edad=28)

zip() -> recibe dos elementos iterables del mismo tamaño y devuelve un diccionario relacionandolos

diccionario = dict(zip(["nombre", "apellido"], ["daniel", "tamargo"]))

items() -> devuelve una lista de tuplas, cada tupla será la clave y el valor (si la clave tenía varios valores, el valor será una tupla de los valores)

items = diccionario.items(9)

keys() y values() -> devuelve una lista de claves o de valores

claves = diccionario.keys()

valores = diccionario.values()

diccionario.clear() -> deja el diccionario vacío

dic1 = diccionario.copy() -> hace una copia del diccionario para trabajarlo de manera independiente

fromkeys() -> declaras las claves y les asignas a todas el mismo valor, si lo dejas vacío le asignará none

diccionario = dict.fromkeys(['a','b','c'], 1)

get() -> es un getter de la clave que le señales

valor = diccionario.get("nombre")

pop() -> recibe una clave, elimina esa clave y su valor del diccionario, y devuelve el valor

valor = diccionario.pop("nombre")

update() -> contrasta un diccionario con otro, si hay claves repetidas, copia el valor del diccionario que le pasamos en el valor del diccionario que usamos el método

dic1.update(dic2)

**Strings**

mi\_cadena = "Hola mundo"

print(mi\_cadena[0:4])

>>> "Hola"

print(mi\_cadena[1:])

>>> "ola Mundo"

print(mi\_cadena[:4])

>>> "Hola"

upper()

lower()

capitalize() -> la primera letra en mayúsculas

swapcase() -> cambia mayúsculas por minúsculas

title() -> la primera letra de cada palabra en mayúsculas

center(30, "=") -> print(texto.center(30, "=")

ljust(30, "=") -> igual que center() pero alineando a la izquierda

rjust(30, "=") -> igual que center() pero alineando a la derecha

zfill(longitud) -> devuelve una copia de la cadena con ceros a la izquierda hasta llegar a la longitud

count(texto)

find("subcadena"[, posicion\_inicial, posicion\_final]) -> busca la subcadena, si la encuentra, devuelve el valor donde empieza esa cadena, si no la encuentra devuelve un -1

startswith("subcadena" [, posicion\_inicio, posicion\_fin])

endswith("subadena" [, posicion\_inicio, posicion\_fin])

isalnum() -> saber si una cadena es alfanumérica

isalpha() -> saber si una cadena es alfabética

isdigit() -> saber si una cadena es numérica

islower() -> saber si una cadena sólo tiene minúsculas

isupper() -> saber si una cadena sólo tiene mayúsculas

isspace() -> saber si una cadena sólo tiene espacios en blanco

istitle() -> saber si una cadena tiene formato de título

replace()

buscar = "Java"

reemplazar="Python"

print ("Estoy estudiando Java".replace(buscar, reemplazar))

>>> Estoy estudiando Python

strip()

mi\_cadena = " Esta es una cadena de texto "

print (mi\_cadena.strip())

>>> Esta es una cadena de texto

lstrip()

mi\_cadena = "www.google.es"

print (mi\_cadena.lstrip("w."))

>>> google.es

format

mi\_cadena = "Hoy es {0} y voy a estudiar {1}"

print (mi\_cadena.format("viernes","Python"))

>>> Hoy es viernes y voy a estudiar Python

split()

mi\_cadena = "Python es divertido"

print(mi\_cadena.split())

>>> ["Python","es","divertido"]

mi\_cadena2 = "Python es, divertido"

print(mi\_cadena2.split(","))

>>> ["Python es","divertido"]

**Comprobaciones**

import re

regex = "^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$"

def check(email):

if(re.search(regex,email)):

print("Valid Email")

else:

print("Invalid Email")

**Ficheros**

import os

**.txt**

r, r+, w, w+, a, a+ (añadir una b para modo binario)

archivo = open(“fichero.txt”, “r”)

contenido = archivo.read() -> read lee todo el contenido

linea1 = archivo.readline() -> readline lee una línea

for linea in archivo: linea = linea.rstrip(‘\n’)

for linea in archivo.readlines(): linea = linea.rstrip(‘\n’)

• closed: devuelve True si el archivo se ha cerrado. De lo contrario, False.

• mode: devuelve el modo de apertura.

• name: devuelve el nombre del archivo

• encoding: devuelve la codificación de caracteres de un archivo de texto

archivo.close()

**.csv (dictReader dictWriter)**

Import csv

With open(“archivo.csv”) as csvfichero:

reader = csv.DictReader(csvfichero)

for fila in reader:

print(fila[‘nombre’], fila[‘apellido])

**dictReader**

resultados = []

with open(‘nombre\_archivo.csv') as csvfichero:

reader = csv.DictReader(csvfichero)

for fila in reader:

resultados.append(fila) -> print (resultados)

**dictWriter**

with open('datos\_ejemplo.csv', 'w') as csvfichero:

campos = ['Nombre', 'Apellido', 'Nota']

writer = csv.DictWriter(csvfichero, fieldnames= campos)

writer.writeheader()

writer.writerow({'Nota': 'B', 'Nombre': 'Aitor', 'Apellido': 'Urrutia'})

(también existe writerows, es pasar una lista de diccionarios, cada línea es un diccionario con claves y valores)

**Fechas** <https://codigofacilito.com/articulos/fechas-python>

from datetime import date

from datetime import datetime

from datetime import timedelta

hoy = date.today() # .day .month .year

ahora = datetime.now() # .day .month .year .hour .minute .second .milisecond

nueva\_fecha = ahora + timedelta(days=2)

formato = ahora.strftime(“Día: %d, Mes: %m, Año: %Y, Hora: %H, Minuto: %M, Segundo: %S”) print(formato)

Colores:

Usando ANSI:

Letras:

print('\033[36m' + 'some marine blue text')

print('\033[35m' + 'some bright magenta text')

print('\033[34m' + 'some blue text')

print('\033[32m' + 'some yellow text')

print('\033[31m' + 'some red text')

print('\033[91m' + 'some bright red text')

print('\033[90m' + 'some bright black text')

print('\033[39m') # and reset to default color

Área:

mismos colores pero desde 41m hasta 49m

print('\033[49m') # and reset to default color