



Ejercicio: Se trata de hacer un proyecto Python “*T06_Número_Combinatorio*” con dos archivos “.py” dentro de la carpeta *src*

- El primer archivo “*combinatoria.py*” contendrá dos funciones:
 - “*factorial*” que recibiendo como parámetro *un número entero* no negativo devuelva el factorial de dicho número.
Recuerda que : $n! = 1 \times 2 \times 3 \times \dots \times (n - 1) \times n$ y que $0! = 1$.
 - “*número_combinatorio*” que recibiendo como parámetro *dos números* no negativos devuelva el número combinatorio.

Recuerda que:
$$\binom{m}{n} = \frac{m!}{n! \times (m-n)!}$$

- El segundo archivo “*test_combinatoria.py*” contendrá las instrucciones necesarias para pedir por teclado dos números y probar la función “*número_combinatorio*”. Se debe comprobar que si el numerador es menor que el denominador visualice directamente “*El numerador es más pequeño que el denominador*”.

- Juego de Ensayo: “ 5 sobre 0 es 1” ; “5 sobre 5 es 1” ;...; “5 sobre 3 es 10”
 $\binom{5}{0} \rightarrow 1$; $\binom{5}{5} \rightarrow 1$; $\binom{5}{1} \rightarrow 5$; $\binom{5}{4} \rightarrow 5$; $\binom{5}{2} \rightarrow 10$; $\binom{5}{3} \rightarrow 10$



Trabajando con cadenas –String- (str)

Son literales que permiten trabajar con textos “alfanuméricos”. También pueden almacenar un único carácter.

Sintaxis:

Se encierran entre apóstrofes, dobles comillas o tres apóstrofes (en este último caso, permite literales de más de una línea).

Ejemplo:

- ‘Esto es un literal, también llamado cadena o String, construido entre apóstrofes’*
- “Esto también es otro literal, salvo que para construirlo se ha usado las dobles comillas”*
- ““Este es también un literal, pero tiene la versatilidad de poder escribirse en más de una línea.*

A la hora de visualizarse por la consola, dado que se ha escrito en cuatro líneas también se visualiza en cuatro líneas ””

- ‘2’, “2”, ““2”” o ‘A’, “a”, ““=”” → también son literales*



Trabajando con cadenas –String- (str)

Algunas operaciones sobre cadenas

Si cadena=“Fundamentos de Programación”

- **len**(cadena) devuelve el número de caracteres que tiene: **27** (incluye espacios en blanco)
- Se accede a una/s posiciones concretas con el operador **[]**

¡Ojo! El primer elemento es el **0**. Así que:

- cadena[0] → **F**
- cadena[1] → **u**
- cadena[-1] → **n** (índices negativos cuentan desde el final)
- cadena[-4] → **c**
- cadena[3:9] → **dament** (observar que no llega a la posición 9. Se queda en la 8)
- cadena[10:-10] → **s de Pr**

Se puede omitir el primero o el segundo de los valores para acceder desde el primero o hasta el último:

- cadena[:9] → **Fundament**
- cadena[20:] → **amación**
- cadena[-5:] → **ación**



Trabajando con cadenas –String- (str)

Operaciones de concatenación

- Operador **+**: concatena las cadenas de su izquierda y derecha:
 - ‘Funda’ + “ment” + ‘os de Pr’ + “ogramación” → *Fundamentos de Programación*
- Operador *****: concatena la cadena tantas veces consigo mismo como indique el número que sigue al operador. Si cadena=“Betis – Sevilla”
 - cadena*3 → *Betis – SevillaBetis – SevillaBetis – Sevilla*



Trabajando con cadenas –String- (str)

Operaciones relacionales para cadenas.

- Operadores relacionales (<, <=, >, >=, ==).

Permiten compara dos cadenas por su léxico:

1. *Se comparan los dos primeros caracteres y una será menor que la otra si su primer carácter está antes que el primer carácter de la otra en el diccionario.*
2. *Si son iguales, se comparan los segundos caracteres y así sucesivamente hasta que encontrar una pareja en una posición en que no sean iguales o una de ellas se haya acabado (en este caso, esta será la menor).*
3. *Si tienen la misma longitud y contienen los mismos caracteres y en el mismo orden son iguales (==)*
 - *'Ana' < 'Alfonso' → False (la ele está antes que la ene en el alfabeto).*
 - *'Ana' < 'alfonso' → True (las mayúsculas están antes que las minúsculas)*



Trabajando con cadenas –String- (str)

Secuencias de escape.

- *Existen algunos caracteres que tienen un significado especial anteponiéndoles \ (son un carácter, aunque se escriban con dos o más pulsaciones de teclado).*

Las más comunes son:

- *\n: Salta una línea*
- *\t: Inserta una tabulación*
- *\': Inserta un apostrofe (') en una cadena encerrada entre apóstrofes ' '*

'Eres un \'listillo\' que te aprovechas de otros' → Eres un 'listillo' que te aprovechas de otros

- *\": Inserta unas comillas (") en una cadena encerrada entre comillas " "*

"Eres un \"listillo\" que te aprovechas de otros" → Eres un "listillo" que te aprovechas de otros



Trabajando con cadenas –String- (str)


Recorrido de cadenas.

Se puede recorrer una cadena mediante un for con las dos siguientes:

Sintaxis:


a) Recorrido como contenedor

```
grado="Ing. Software"  
for letra in grado:  
    print(letra)
```



b) Recorrido por posición

```
grado="Ing. Software"  
for posición in range(0,len(grado)):  
    print(grado[posición])
```



I
n
g
.
S
o
f
t
w
a
r
e



Sentencia break

Permite finalizar anticipadamente la ejecución de un for o un while (debe estar dentro de un if)

Sintaxis :

for **variable** **in** contenedor:

if condición:

...
break
sentencias-n

...
sentencia-m

Se dispone de una frase, *mi_frase*="Fundamentos de Programación" y se quiere contar cuántos caracteres hay hasta encontrar el primer espacio en blanco:

contador=0

for **carácter** **in** *mi_frase*:

if *carácter*==" ":

break

contador=*contador*+1

print ('El contador vale:', *contador*)

→ El contador vale: 11



Ejercicio: Se trata de hacer un proyecto “*T07_Palíndromo*” con dos archivos “.py”

- El primer módulo “*cadena.py*” contendrá una función “*es_palíndromo*” que recibiendo una cadena debe devolver *True* o *False* según la cadena sea, o no, un palíndromo.
- El segundo módulo “*test_cadena.py*” contendrá las instrucciones necesarias para pedir una palabra (o una frase) y visualizar si lo que se ha introducido es, o no, un palíndromo.

Notas.

- Una cadena es un palíndromo si se lee igual de derecha a izquierda como de izquierda a derecha.
- Recuerde que:
 - La función *range* devuelve secuencias de números.
 - *len(cadena)* devuelve la longitud de una cadena.
 - El operador aritmético *//* devuelve la parte entera de una división.



Contenedores

Los contenedores son *estructura de datos* que permiten, por su tipología, semejanza, funcionalidad, etc., agruparlos bajo un único tipo de datos: Estos tipos pueden ser en Python:

- Tuplas → (...)
- Listas → [...]
- Conjuntos → {...}
- Diccionarios → {.: ..., .: ...}



Contenedores: tuple (tupla)

Variable que, generalmente, se usa almacena más de un dato sobre un mismo objeto y referenciarlos bajo una misma denominación.

- Sintaxis: se encierran entre paréntesis y separados por coma los valores del objeto en cuestión.

- Por ejemplo:

tuplas con datos de personas –con 6 datos por persona-:

- ('12345678A', 'Arancha', 'López Martín', 18, 'Sevilla', 'Sevilla')
- ('111222333B', 'Antonio', 'Gómez Benítez', 19, 'Lepe', 'Huelva')

tuplas con datos sobre admisión a grados –con 4 datos por grado-:

- ('Ingeniería del Software', 11.184, 174, '814502')
- ('Tecnología Informática', 9,941, 124, '823504')

- Se accede a los elementos de una tupla añadiendo tras el nombre de la tupla y entre corchetes [] la posición en la tupla (*¡OJO! Se empieza por 0*). (*Desde el final con -1*)
- **Las tuplas son inmutables** → Una vez creada una tupla, no se puede añadir, suprimir o modificar sus datos. Es necesario, generar una nueva a partir de sus datos.



Contenedores: **tuple (tupla)**

Ejemplo de acceso a los elementos de una tupla

Si se definen de las siguientes tuplas:

- persona1=('12345678A', 'Arancha', 'López Martín', 'M', 18, 'Sevilla', 'Sevilla')
- persona2=('111222333B', 'Antonio', 'Gómez Benítez', 19, 'Lepe', 'Huelva')
- grado1=('Ingeniería del Software', 11.184, 174, '814502')
- grado2=('Tecnología Informática', 9,941, 124, '823504')

Se accede a sus elementos:

- persona1[1]→ 'Arancha' ; persona1[0]→ '12345678A' ; persona2[5] →'Huelva'
- grado1[1]→ 11,184 ; grado1[3]→'814502' ; también grado1[-1]→'814502'
- grado2[2]=180 --- Proporciona un **error** de compilación (*inmutabilidad de las tuplas*)

Acceso a un carácter de una cadena que es un elemento de una tupla:

(de los más general a lo más particular)

- persona2[2][6]→'B' ; persona2[-1][-1]→'a'



Tuplas con nombres (namedtuple –contrucción de la tupla-)

*Para no tener que usar referencias indexadas (los corchetes y la posición) para acceder a los elementos de una tupla se usa la función **namedtuple** que asigna un nombre descriptivo a cada campo de una tupla antes de su creación. Se accede con el operador punto (.) y el nombre.*

NOTA:-→ Es necesario importar la función desde la biblioteca *collections*
`from collections import namedtuple`

Sintaxis:

Nombre de la tupla=**namedtuple**("nombre", "nombre campo1, nombre campo2, nombre campo3, ...")

*¡Ojo!: **namedtuple** sólo tiene **2 parámetros** separados por una coma. El segundo parámetro es una cadena que contiene los nombres de los campos, a su vez, separados por coma.*



Tuplas con nombres (namedtuple –contrucción de la tupla-)

Sintaxis:

Nombre de la tupla=**namedtuple**("nombre", "nombre campo1, nombre campo2, nombre campo3, ...")

Por ejemplo: Se definen los tipos *Persona* y *Grado*

- *Persona*=**namedtuple**('persona', 'dni, nombre, apellidos, edad, localidad, provincia')
- *Grado*=**namedtuple**('grado', 'nombre, nota_corte, plazas, código')

Cuando se crean la tupla se antepone al paréntesis que abre en nombre de la tupla (nombre del tipo)

- *persona1*=*Persona*('12345678A', 'Arancha', 'López Martín', 18, 'Sevilla', 'Sevilla')
- *persona2*=*Persona*('111222333B', 'Antonio', 'Gómez Benítez', 19, 'Lepe', 'Huelva')
- *grado1*=*Grado*('Ingeniería del Software', 11.184, 174, '814502'),
- *grado2*=*Grado*('Tecnología Informática', 9.941, 124, '823504')

Acceso a los campos de la tupla:

- *persona1.nombre* → 'Arancha' ; *persona1.dni* → '12345678A' ; *persona2.provincia* → 'Huelva'
- *Grado1.nota_corte* → 11,184 ; *grado1.código* → '814502'



Contenedores: “list()” (lista)

- Variable que almacena todo tipo de datos, en un orden determinado.
Por ejemplo: una lista de números enteros, una lista de nombre de ciudades o una lista de tuplas que contienen determinados datos sobre equipos de futbol,...

Sintaxis para crear una lista: con la función *list()* o con corchetes *[]* y, en su caso, con los elementos separado por coma:

Ejemplos:

- lista_vacía = *list()*
- lista_vacía2 = *[]*
- edades = *[23,17,21,17,30,23,11,7]*
- grados = *[Grado('Ingeniería del Software', 11.184, 174, '814502'), Grado('Tecnología Informática', 9.941, 124, '823504'), Grado('Ingeniería de Computadores, 9.425, 102, '815001'), Grado(...),...]*
- *Es muy normal utilizar una lista almacenar las tuplas que forman los registros de un fichero.*



Contenedores: “list()” (lista)

Las listas son *mutables*: se puede añadir, eliminar un dato o modificarlo

- Se *añade* un *dato* al *final* de una lista con `nombre_lista.append(dato)`
- Se *inserta* un *dato* en una *posición* de una lista con `nombre_lista.insert(posición, dato)`
- Se *borra* un *dato* de una lista con `nombre_lista.remove(dato)`
- Se *borra* un dato de una *posición* de una lista con `nombre_lista.pop(posición)`
- Se *accede* a un dato en determinada posición con `nombre_lista[posición]`. ¡Empieza en cero!
- Se *conoce el número de elementos* de una lista con `len(nombre_lista)`

Ejemplo de acceso a un elemento:

`edades=[23,17,21,17,30,23,11,7]`

`grados=[Grado('Ingeniería del Software', 11.184, 174, '814502'), Grado('Tecnología Informática', 9.941, 124, '823504'), Grado('Ingeniería de Computadores', 9.425, 102, '815001'), Grado(...),...]`

- `edades[1] → 17` ; `edades[-2] → 11`
- `grados[1][1]` o `grados[1].nota_corte → 9.941`
- `grados[2][0]` o `grados[2].nombre → 'Ingeniería de Computadores'`




Sentencia for (recorriendo una lista)

Permite ejecutar de forma iterada un bloque de sentencias desde el primero, hasta el último, de los elementos de una lista

Sintaxis :

for *variable* *in* nombre_lista:
 bloque sentencias



La *variable* va tomando, en cada iteración, el valor de cada elemento del contenedor

Ejemplo:

Se dispone de una lista de números *mi_lista*=**[62,56,90,-22,20]**

for *v* *in* *mi_lista*:
 print (*v*+1)



63
57
91
-21
21



Sentencia for (recorriendo una lista)

También se puede recorrer una lista por la posición de sus elementos

Sintaxis :


```
for pos in range(0,len(nombre_lista)):  
    bloque sentencias
```

La variable **pos** va tomando, en cada iteración, el valor que devuelve la secuencia

Ejemplo:

Se dispone de una lista de números **mi_lista=[62,56,90,-22,20]**

```
for pos in range(0,len(mi_lista)):  
    print (mi_lista[pos]+1)
```



63
57
91
-21
21



Sentencia for (recorriendo una lista)

Ejemplo:

Se dispone de una lista de tuplas de tipo Grado:

```
grados = [Grado('Ingeniería del Software', 11.184, 174, '814502'), Grado('Tecnología Informática', 9.941, 124, '823504'), Grado('Ingeniería de Computadores', 9.425, 102, '815001'), Grado(...),...]
```

```
for g in grados:  
    print (g)
```

grado(nombre='Ingeniería del Software', nota_corte=11.184,plazas=174,código='814502')
grado(nombre='Tecnología Informática', nota_corte=9.941,plazas=124,código='823504')
grado(nombre='Ingeniería de Computadores', nota_corte=9.425,plazas=102,código='815001')

```
for nom, not, pl, código in grados:  
    print (nom, ': ', pl)
```

'Ingeniería del Software' : 174
'Tecnología Informática' : 124
'Ingeniería de Computadores' : 102

```
for g in grados:  
    print (g.nombre, ': ', g.plazas)
```



Ejercicio: Se trata de hacer un proyecto “*T08_Datos_Personales*” con dos archivos:

- El primer archivo/módulo *datos_personales.py* contendrá dos funciones
 - “*filtra_por_edad*” que recibiendo como parámetros una lista de tuplas con los datos de personas y una edad, devuelva otra lista con las tuplas de las personas con menos edad que la dada.
 - “*obtiene_dni_y_nombres*” que recibiendo como parámetro una lista de tuplas con los datos de personas, devuelva otra lista con los nombre y los dni de todas la personas.
- El segundo archivo/módulo *test_datos_personales.py* contendrá las instrucciones necesarias para crear una lista y probar las dos funciones. Para crear la lista use los siguientes datos.(tenga cuidado al copiar de ver como quedan los apóstrofes)

Importante: Use el siguiente tipo:

Persona=*namedtuple*('persona', 'dni, nombre, apellidos, edad, localidad, provincia')

```
('12345678A', 'JUAN', 'AFAN POSTIGO', 22, 'SEVILLA', 'SEVILLA')
('12345678B', 'NICOLAS', 'AGUILAR SAUCEDO', 20, 'DOS HERMANAS', 'SEVILLA')
('12345678C', 'LUCAS', 'ACEJO GARCÍA', 20, 'UTRERA', 'SEVILLA')
('12345678D', 'CLAUDIA', 'ÁLVAREZ GARCÍA', 21, 'VISO DEL ALCOR', 'SEVILLA')
('12345678E', 'PAULA', 'ALBENDÍN CAMINO', 19, 'TOMARES', 'SEVILLA')
('12345678F', 'ANA', 'LOBATO ÁLVAREZ', 18, 'PUNTA UMBRÍA', 'HUELVA')
('12345678G', 'ANTONIO', 'DÍAZ NARANJO', 18, 'CHIPIONA', 'CADIZ')
('12345678H', 'SOFÍA', 'GUERRERO CANTARERO', 20, 'CHIPIONA', 'CADIZ')
```