
UAP

UAP | UNIVERSIDAD ADVENTISTA DEL PLATA

ALGORITMOS Y ESTRUCTURAS DE DATOS

CONTROL DE FLUJO – ARRAYS - RECURSIVIDAD

CULTURA

API

Interfaz de programación de aplicaciones: conjunto de clases y funciones prefabricadas que ayudan a simplificar el desarrollo y comunicación de sistemas software.

CULTURA

Referencia y biblioteca del lenguaje

Especificación técnica del modus operandi de un lenguaje. En el caso de Python se puede ver en

- https://docs.python.org/es/3/reference/
- https://docs.python.org/es/3/library/

string — Operaciones comunes de cadena de caracteres

Source code: Lib/string.py

Ver también: Cadenas de caracteres — str

Métodos de las cadenas de caracteres

Constantes de cadenas

Las constantes definidas en este módulo son:

`string.`**`ascii_letters`**
La concatenación de las constantes abajo descriptas `ascii_lowercase` y `ascii_uppercase`. Este valor es independiente de la configuración regional.

`string.`**`ascii_lowercase`**
Las letras minúsculas `'abcdefghijklmnopqrstuvwxyz'`. Este valor es independiente de la configuración regional y no cambiará.

`string.`**`ascii_uppercase`** ¶
Las letras mayúsculas `'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`. Este valor es independiente de la configuración regional y no cambiará.

`string.`**`digits`**
La cadena `'0123456789'`.

`string.`**`hexdigits`**
La cadena `'0123456789abcdefABCDEF'`.

`string.`**`octdigits`**
La cadena de caracteres `'01234567'`.

https://learnxinyminutes.com/docs/python/

Principales conceptos

CONCEPTO

IF, ELIF, ELSE

Dependiendo del resultado de una evalución lógica ejecuta un bloque de código o no

```
1 x = int(input('Escriba un nro: '))
2 if x < 0:
3     print('es negativo')
4 elif x == 0:
5     print('es cero')
6 else:
7     print('es positivo')
8
```

CONCEPTO

CORTOCIRCUITO LÓGICO (repaso)

Método para evitar evaluar todos los términos de una expresión lógica.

if False and edad < 99 and len(nombre) > 50

CONCEPTO

SWITCH

```
1 tipo = 'coche'
2 if tipo == 'coche':
3     ruedas = 4
4 elif tipo == 'bicicleta':
5     rueda = 2
6 elif tipo == 'camión':
7     rueda = 6
8 else:
9     rueda = 0
10
```

CONCEPTO

IF TERNARIO

Forma simplificada de la sentencia if, solo cuando se desea usar una sola sentencia, no un bloque de código.

```python
edad = 55
categoria = 'Cadete' if edad < 15 else 'Adulto'
print(categoria)
```

CONCEPTO

VARIABLE CONTADOR

Estructura de incrementar o decrementar el contenido de una variable de forma constante:

cont = cont + 1

cont += 1

x = x – 3

x -= 3

CONCEPTO

VARIABLE ACUMULADOR

Estructura de incrementar o decrementar el contenido de una variable de forma variable:

acu = acu + x #acu += x

acu = acu – x #acu -= x

CONCEPTO

WHILE, ELSE

Mecanismo para generar una iteración o un bucle. Repite un bloque de código mientras el condicional sea verdadero.

Tener cuidado de formar un bucle infinito.

```python
c = 0
while(c < 10):
    print(c)
    c = c + 1
else:
    print("pasa por aca cuando el condicional es false")
```

CONCEPTO

FOR, ELSE

Produce itaraciones recorriendo listas o rangos.

```python
#recorrer lista
for c in ['a', 'b', 'c']:
    print(c)
else:
    print("pasa por aca no hay más para iterar")

#recorrer cada letra de la frase
for s in 'Es contra el viento que la cometa se eleva. W. S. Churchill':
    print(s)

#un rango de números
for i in range(10):
    print(i)

#un rango de números desde, hasta y paso
for i in range(5, 20, 2):
    print(i)

#recorrer lista con índice
animals = ["dog", "cat", "mouse"]
for i, value in enumerate(animals):
    print(i, value)
```

No cambiar la cantidad de elementos de un array cuando se usa con un for.
De necesitar agregar o quitar trabajar con una copia.

CONCEPTO

BREAK, CONTINUE

Forma de cortar un bucle (break) o forzar al siguiente ciclo normalmente saltando un bloque de código.

```
#un rango de números
for i in range(10):
    if i > 7:
        break
    if i == 3:
        continue
    print(i)
"""
0
1
2
4
5
6
7
"""
```

CONCEPTO

SECUENCIAS: Listas, Tuplas, Rangos

```python
#Constructor de una lista
a = list(['a', 1, True])

#contrucción de forma abreviada
a = ['a', 1, True]

print(a)
#['a', 1, True]

print(a[0])
#a

print(type(a))
#<class 'list'>

#ver todo lo que puedo hacer con una lista
print(dir(list()))
```

```
#ver todo lo que puedo hacer con una lista
print(dir(list()))
"""
<class 'list'>
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__',
'__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__',
 '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
 '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
 '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__',
 '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend',
 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
"""
```

```
1 #Constructor de una matriz
2 m = [
3       [4, 2, -1],
4       [0, -3, 0],
5       [1, 6, 8]
6 ]
7
8 print(m)
```

```python
# Tuplas son listas pero inmutables
tup = (1, 2, 3)
tup[0]          # => 1
tup[0] = 3  # Raises un TypeError
```

```python
li = [1, 2, 4, 3]

# Acceso a un elemento
li[0]   # => 1
# Acceso pero el final de la lista
li[-1]  # => 3

# Si se quiere acceder a un elemento que no existe
li[4]  # no da un error del tipo IndexError

# Se puede acceder a una porción de la lista: slices
li[1:3]   # nos devuelve el index 1 al 3 => [2, 4]
li[2:]    # nos devuelve desde el index 2 hasta el final => [4, 3]
li[:3]    # nos devuelve desde el inicio hasta el index 3  => [1, 2, 4]
li[::2]   # nos devuelve una lista elemento por medio => [1, 4]
li[::-1]  # invirte el orden de la lista => [3, 4, 2, 1]
# Sintáxis general de los slices
# li[start:end:step]

# Otra forma de hacer una copia de la lista
li2 = li[:]  # => li2 = [1, 2, 4, 3] pero (li2 is li) resultará false.
```

```python
#Tipo particular de secuencias de números enteros inmutables
inicio, final, paso = 0, 10, 2
a = range(final)
print(a)
#range(0, 10)

a = range(inicio, final, paso)
print(a)
#range(0, 10, 2)
```

CONCEPTO

CONJUNTOS (set)

Colección no ordenada de eleméntos únicos.

```ruby
#Set: conjunto de elementos únicos
filled_set = {1, 1, 2, 2, 3, 4, 5}
#{1, 2, 3, 4, 5}

other_set = {3, 4, 5, 6}

#intersección &
filled_set & other_set  # => {3, 4, 5}

# Unión |
filled_set | other_set  # => {1, 2, 3, 4, 5, 6}

# Diferencia -
{1, 2, 3, 4} - {2, 3, 5}  # => {1, 4}

# Diferencia simétrica ^
{1, 2, 3, 4} ^ {2, 3, 5}  # => {1, 4, 5}

# Comparación por un superset
{1, 2} >= {1, 2, 3} # => False

# Comparación por un subset
{1, 2} <= {1, 2, 3} # => True

# Buscar en el conjunto
2 in filled_set   # => True
10 in filled_set  # => False
```

CONCEPTO

MAPAS (diccionarios)

Estructura contenedera de elementos en pares clave => valor. Las claves deben ser únicas.

```python
# Construcción de un diccionario en pares clave=>valor
empty_dict = {} #contrucción vacío
filled_dict = {"one": 1, "two": 2, "three": 3}

print(filled_dict["two"])
#2

for k, v in filled_dict.items():
    print("clave: " + k + " - valor: " + str(v))
#clave: one - valor: 1
#clave: two - valor: 2
#clave: three - valor: 3
```

```
14 for k in filled_dict.keys():
15     print("clave: " + k)
16 #clave: one
17 #clave: two
18 #clave: three
19
20 for v in filled_dict.values():
21     print("valor: " + str(v))
22 #valor: 1
23 #valor: 2
24 #valor: 3
```

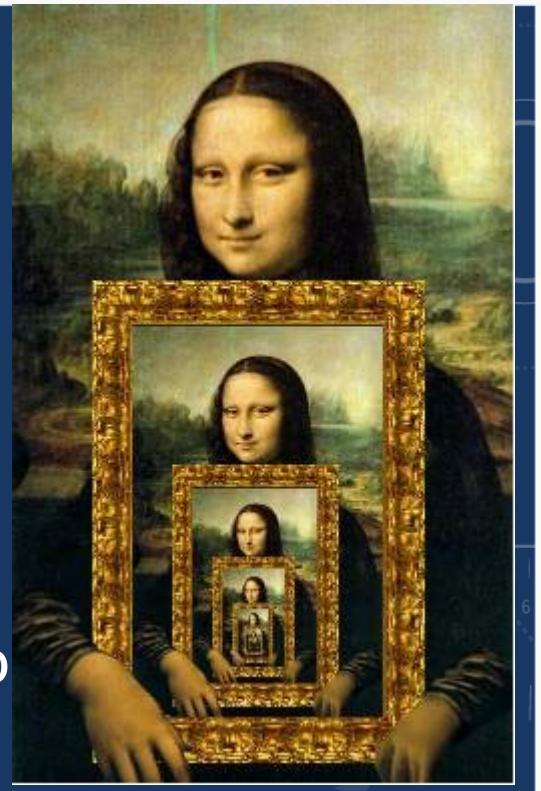
```python
# buscando un key
"one" in filled_dict  # => True
1 in filled_dict      # => False

# Agregando elemento
filled_dict.update({"four":4})  # => {"one": 1, "two": 2, "three": 3, "four": 4}
filled_dict["four"] = 4         # alternativa

# Sacando elemento
del filled_dict["one"]
```

CONCEPTO

RECURSIVIDAD



Forma de iterar en donde una función se llama a sí misma.

También hay que tener cuidado de no llegar a un bucle infinito

Una función recursiva siempre tendrá:

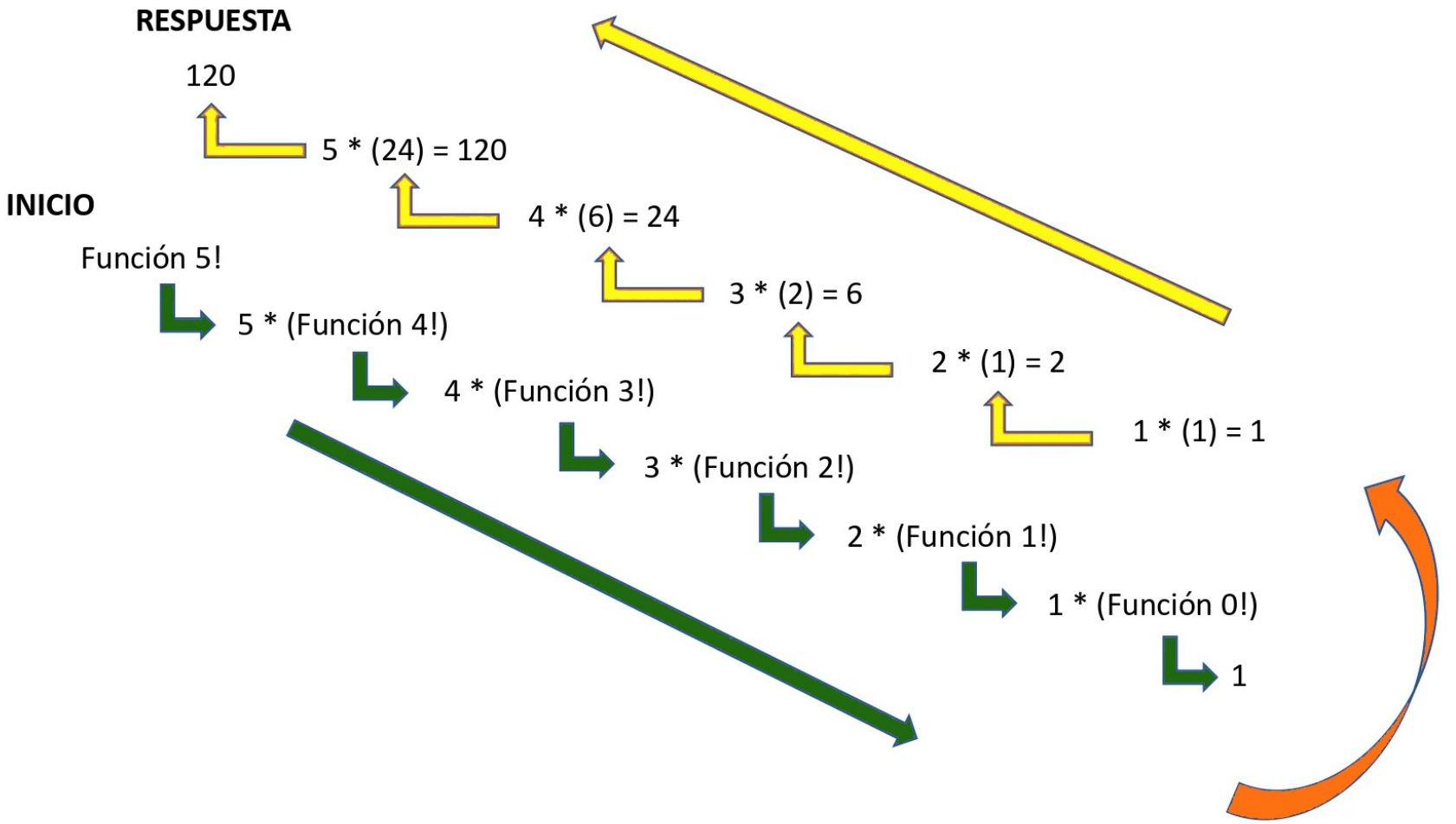
- Al menos una llamada a sí misma, sino no es recursiva
- Al menos un if con el caso base que corta la recursividad
- Python tiene un límite de 1000 llamados recursivos de profundidad

```
77 n = 5
78 def test(e):
79     print(id(e))
80     print(e)
81
82 print(id(n))
83 test(n)
84 #139864080042416
85 #139864080042416
86 #5
```

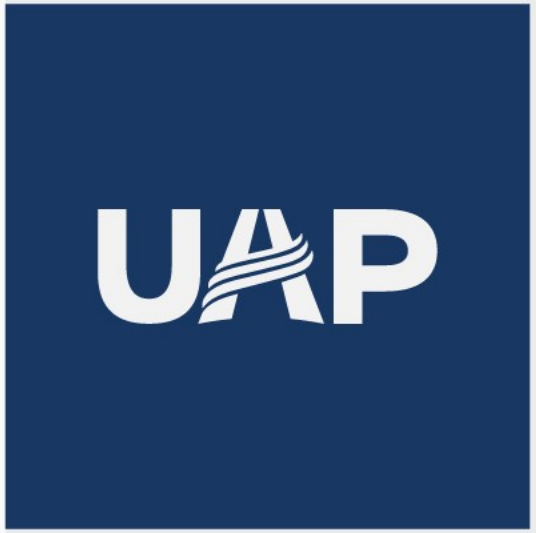
```
77 n = 5
78 def test(e):
79     print(id(e))
80     print(e)
81
82 print(id(n))
83 test(n + 1)
84 #139864080042416
85 #140498654530000
86 #6
```

```python
def factorial(x):
    print(id(x))
    if x < 1:
        return 1 #caso base
    else:
        return x * factorial(x-1)

n = 5# !5 = 5 * 4 * 3 * 2 * 1 = 120
print("Factorial de " + str(n) + " = " + str(factorial(n)))
#140559147362736
#140559147362704
#140559147362672
#140559147362640
#140559147362608
#140573109008656
#Factorial de 5 = 120
```


RESPUESTA
120
5 * (24) = 120
4 * (6) = 24
3 * (2) = 6
2 * (1) = 2
1 * (1) = 1
INICIO
Función 5!
5 * (Función 4!)
4 * (Función 3!)
3 * (Función 2!)
2 * (Función 1!)
1 * (Función 0!)
1

```
40 def sumatoria(n, c = 0):
41     if c >= n:
42         return c
43     else:
44         return c + sumatoria(n, c + 1)
45
46 print("Sumatoria: " + str(sumatoria(5)))
47 #-------------------------------------
48 c = 0
49 n = 5
50 t = 0
51 while(c < n):
52     c = c + 1
53     t = t + c
54
55 print("Fin while: " + str(t))
```



UAP.EDU.AR | @UAPARGENTINA