# Laboratorio: Programación en Python

En este laboratorio, pondremos en práctica algunos de los conceptos que hemos aprendido recientemente. El objetivo es que implementes cada función utilizando tres enfoques diferentes en Python: **programación imperativa**, **programación orientada a objetos** y **programación funcional**.

> **NOTA:** En este laboratorio deberías intentar escribir todas las funciones por ti mismo utilizando únicamente la sintaxis más básica de Python, sin utilizar funciones predefinidas como `len`, `count`, `sum`, `max`, `min`, `in`, etc. ¡Dale una oportunidad!

La celda después de cada ejercicio contiene pruebas para verificar si tu función funciona como se espera.

```
from mod.testing import *
import unittest
```

## 1. Escribe una función que devuelva el mayor de dos números

```
#tu codigo aquí
def greater_imperativa(a, b):
    if a > b:
        return a
    else:
        return b

# Para testear tu función
test_greater(greater_imperativa)
```

```
.....................................................................
.............................
----------------------------------------------------------------------
Ran 100 tests in 0.204s

OK
```

```
class Comparador:
    @staticmethod
    def greater_poo(a, b):
        return a if a > b else b
greater_poo = Comparador.greater_poo

test_greater(greater_poo)
```

```
.....................................................................
.............................
```

```
----------------------------------------------------------------------
Ran 100 tests in 0.169s

OK
```

```
greater_funcional = (lambda a, b: a if a > b else b)

test_greater(greater_funcional)
```

```
....................................................................
............................
----------------------------------------------------------------------
Ran 100 tests in 0.162s

OK
```

## 2. Ahora escribe una función que devuelva el elemento más grande de una lista numerica

```
#tu codigo aquí

def greatest_imperativa(arr):
    try:
        max_num = arr[0]
    except IndexError:
        return None
    index = 1
    while True:
        try:
            max_num = greater_imperativa(arr[index],max_num)
            index += 1
        except IndexError:
            break

    return max_num

# Para testear tu función
test_greatest(greatest_imperativa)
```

```
....................................................................
............................
----------------------------------------------------------------------
Ran 100 tests in 0.166s

OK
```

```
class ComparadorListas:
    @staticmethod
    def greatest_poo(arr):
        try:
```

```
            max_num = arr[0]
        except IndexError:
            return None

        return ComparadorListas.recorrer_lista(arr, max_num, 1)

    @staticmethod
    def recorrer_lista(arr, max_num, index):
        while True:
            try:
                max_num = greater_poo(arr[index], max_num)
                index += 1
            except IndexError:
                break
        return max_num


greatest_poo = ComparadorListas.greatest_poo

test_greatest(greatest_poo)
```

```
.....................................................................
.............................
----------------------------------------------------------------------
Ran 100 tests in 0.149s

OK
```

```
greatest_funcional = lambda arr: None if arr == [] else
reduce(greater_funcional, arr)

test_greatest(greatest_funcional)
```

```
.....................................................................
.............................
----------------------------------------------------------------------
Ran 100 tests in 0.172s

OK
```

## 3. Escribe una función que sume todos los elementos de una lista

```
def sum_imperativa(arr):
    total=0
    index=0
    while True:
        try:
            total+=arr[index]
            index+=1
```

```
        except IndexError:
            break
    return total

# Para testear tu función
test_sum(sum_imperativa)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.212s

OK
```

```
class Sumador:
    @staticmethod
    def sum_poo(arr):
        return Sumador.recorrer_lista(arr, 0, 0)

    @staticmethod
    def recorrer_lista(arr, total, index):
        while True:
            try:
                total += arr[index]
                index += 1
            except IndexError:
                break
        return total


sum_poo = Sumador.sum_poo

test_sum(sum_poo)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.179s

OK
```

```
sum_funcional = lambda arr: 0 if arr == [] else reduce(lambda a, b: a
+ b, arr)

test_sum(sum_funcional)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.166s

OK
```

## 4. Escribe otra función que multiplique todos los elementos de una lista

```
def mult_imperativa(arr):
    if arr == []:
        return 0
    product = 1
    index = 0
    while True:
        try:
            product *= arr[index]
            index += 1
        except IndexError:
            break
    return product

# Para testear tu función
test_mult(mult_imperativa)
```

```
..........................................................................
.............................
----------------------------------------------------------------------
Ran 100 tests in 0.166s

OK
```

```
class Multiplicador:
    @staticmethod
    def mult_poo(arr):
        if arr == []:
            return 0
        return Multiplicador.recorrer_lista(arr, 1, 0)

    @staticmethod
    def recorrer_lista(arr, product, index):
        while True:
            try:
                product *= arr[index]
                index += 1
            except IndexError:
                break
        return product


mult_poo = Multiplicador.mult_poo

test_mult(mult_poo)
```

```
..........................................................................
.............................
```

```
----------------------------------------------------------------------
Ran 100 tests in 0.139s

OK
```

```
mult_funcional = lambda arr: reduce(lambda a, b: a * b, arr, 1) if arr
!= [] else 0

test_mult(mult_funcional)
```

```
.....................................................................
.............................
----------------------------------------------------------------------
Ran 100 tests in 0.133s

OK
```

## 5. Ahora combina esas dos ideas y escribe una función que reciba una lista y ya sea "+" o "*", y produzca el resultado acorde

```
def operaciones_imperativa(arr, op ='+'):
    if arr == []:
        return 0

    if op == "+":
        return sum_imperativa(arr)
    elif op == "*":
        return mult_imperativa(arr)
    else:
        raise ValueError("Operación no soportada")

# Para testear tu función
test_operations(operaciones_imperativa)
```

```
.....................................................................
.............................
----------------------------------------------------------------------
Ran 100 tests in 0.147s

OK
```

```
class Calculadora:
    @staticmethod
    def operaciones_poo(arr, op):
        if arr == []:
            return 0
        return Calculadora.ejecutar_operacion(arr, op)
```

```
    @staticmethod
    def ejecutar_operacion(arr, op):
        if op == "+":
            return Sumador.sum_poo(arr)
        elif op == "*":
            return Multiplicador.mult_poo(arr)
        else:
            raise ValueError("Operación no soportada")

operaciones_poo = Calculadora.operaciones_poo

test_operations(operaciones_poo)
```

```
..........................................................................
.............................
----------------------------------------------------------------------
Ran 100 tests in 0.152s

OK
```

```
operaciones_funcional = lambda arr, op: (0 if arr == [] else
                                         sum_funcional(arr) if op ==
"+" else
                                         mult_funcional(arr) if op ==
"*" else
                                         ValueError("Operación no
soportada"))

test_operations(operaciones_funcional)
```

```
..........................................................................
.............................
----------------------------------------------------------------------
Ran 100 tests in 0.150s

OK
```

## 6. Escribe una función que devuelva el factorial de un número.

```
# Fórmula factorial
# n! = n * (n - 1) *...*1

# Este código define una función llamada "factorial" que toma una
entrada "n". La función utiliza un bucle for para iterar a través del
rango de números
# desde 1 hasta n+1. Para cada número en ese rango, multiplica el
valor actual de x por el número en el rango. Al final del bucle,
# la función devuelve el valor final de x, que será el factorial del
número de entrada "n".
```

```python
# El factorial de un entero positivo n es el producto de todos los
enteros positivos menores o iguales a n.
# Por ejemplo, el factorial de 6 (escrito "6!") es 6 * 5 * 4 * 3 * 2 *
1 = 720.

# Así que esta función toma una entrada de cualquier entero positivo y
devuelve el factorial de ese número.

def factorial_imperativa(n):
    if n < 0:
        raise ValueError("El factorial no está definido para números
negativos.")
    if n == 0 or n == 1:
        return 1
    return n * factorial_imperativa(n - 1)

# Para testear tu función
test_factorial(factorial_imperativa)
```

```
.....................................................................
.............................
----------------------------------------------------------------------
Ran 100 tests in 0.148s

OK
```

```python
class CalculadoraFactorial:
    @staticmethod
    def factorial_poo(n):
        if n < 0:
            raise ValueError("El factorial no está definido para
números negativos.")
        if n == 0 or n == 1:
            return 1
        return n * CalculadoraFactorial.factorial_poo(n - 1)

factorial_poo = CalculadoraFactorial.factorial_poo

test_factorial(factorial_poo)
```

```
.....................................................................
.............................
----------------------------------------------------------------------
Ran 100 tests in 0.141s

OK
```

```python
factorial_funcional = lambda n: (ValueError("El factorial no está
definido para números negativos.")
                                 if n < 0 else
```

```
                                    (1 if n == 0 else n *
factorial_funcional(n - 1)))

test_factorial(factorial_funcional)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.154s

OK
```

## 7. Escribe una función que tome una lista y devuelva una lista de los valores únicos.

`NOTE: No podemos usar set. `

```python
def unique_imperativa(arr):
    unique_list = []
    index = 0

    while True:
        try:
            item = arr[index]
            is_unique = True
            unique_index = 0

            while True:
                try:
                    if unique_list[unique_index] == item:
                        is_unique = False
                        break
                    unique_index += 1
                except IndexError:
                    break

            if is_unique:
                unique_list.append(item)

            index += 1

        except IndexError:
            break

    return unique_list

# Para testear tu función
test_unique(unique_imperativa)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.406s

OK
```

```python
class UniqueFinder:
    @staticmethod
    def unique_poo(arr):
        unique_list = []
        index = 0

        while True:
            try:
                item = arr[index]
                if UniqueFinder.is_unique(item, unique_list):
                    unique_list.append(item)
                index += 1

            except IndexError:
                break

        return unique_list

    @staticmethod
    def is_unique(item, unique_list):
        unique_index = 0

        while True:
            try:
                if unique_list[unique_index] == item:
                    return False
                unique_index += 1
            except IndexError:
                break

        return True

unique_poo = UniqueFinder.unique_poo

test_unique(unique_poo)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.402s

OK
```

```
unique_funcional = lambda arr: reduce(
    lambda acc, item: acc + [item] if reduce(
        lambda x, y: x or (y == item), acc, False
    ) == False else acc, arr,[]
)

test_unique(unique_funcional)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.670s

OK
```

## 8. Escribe una función que devuelva la moda de una lista, es decir: el elemento que aparece más veces.

`NOTE: No se debe usar count... `

```
def moda_imperativa(arr):
    unique_list = unique_imperativa(arr)
    max_frecuencia = -1
    moda = None

    unique_index = 0
    while True:
        try:
            num = unique_list[unique_index]
            frecuencia = 0
            index = 0

            while True:
                try:
                    if arr[index] == num:
                        frecuencia += 1
                    index += 1
                except IndexError:
                    break

            if frecuencia > max_frecuencia:
                max_frecuencia = frecuencia
                moda = num

            unique_index += 1
        except IndexError:
            break

    return moda
```

```
# Para testear tu función
test_mode(moda_imperativa)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.193s

OK
```

```
class CalculadoraModa:
    @staticmethod
    def moda_poo(arr):
        unique_list = unique_poo(arr)
        max_frecuencia = -1
        moda = None
        unique_index = 0

        while True:
            try:
                num = unique_list[unique_index]
                frecuencia = CalculadoraModa.contar_frecuencia(arr,
num)

                if frecuencia > max_frecuencia:
                    max_frecuencia = frecuencia
                    moda = num

                unique_index += 1
            except IndexError:
                break

        return moda

    @staticmethod
    def contar_frecuencia(arr, num):
        frecuencia = 0
        index = 0

        while True:
            try:
                if arr[index] == num:
                    frecuencia += 1
                index += 1
            except IndexError:
                break

        return frecuencia

moda_poo = CalculadoraModa.moda_poo
```

```
test_mode(moda_poo)
```

```
..................................................................
..............................
----------------------------------------------------------------------
Ran 100 tests in 0.196s

OK
```

```
def moda_funcional(arr):

    unique_items = unique_funcional(arr)

    count_occurrences = lambda item, arr: reduce(
        lambda count, x: count + 1 if x == item else count, arr, 0
    )

    frequency = reduce(
        lambda acc, item: acc + [(item, count_occurrences(item,
arr))],
        unique_items,
        []
    )

    moda = reduce(
        lambda acc, item: item if item[1] > acc[1] else acc,
frequency, (None, 0)
    )

    return moda[0] if moda[0] is not None else None

test_mode(moda_funcional)
```

```
..................................................................
..............................
----------------------------------------------------------------------
Ran 100 tests in 0.203s

OK
```

## 9. Escribe una función que calcule la desviación estándar de una lista.

`NOTE: no utilices librerías ni ninguna función ya construida. ☺`

```
def st_dev_imperativa(arr):
    if arr == []:
        raise ValueError("La lista no puede estar vacía.")
```

```
    sum_total = sum_imperativa(arr)
    count = 0

    while True:
        try:
            _ = arr[count]
            count += 1
        except IndexError:
            break

    mean = sum_total / count

    variance_sum = 0
    index = 0

    while True:
        try:
            num = arr[index]
            variance_sum += (num - mean) ** 2
            index += 1
        except IndexError:
            break

    variance = variance_sum / (count - 1) if count > 1 else 0
    return variance ** 0.5

# Para testear tu función
test_stdev(st_dev_imperativa)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.171s

OK
```

```
class StatsCalculator:
    @staticmethod
    def st_dev_poo(arr):
        if arr == []:
            raise ValueError("La lista no puede estar vacía.")

        sum_total = sum_poo(arr)
        count = StatsCalculator.contar_elementos(arr)
        mean = sum_total / count
        variance_sum = StatsCalculator.calcular_varianza(arr, mean)
        variance = variance_sum / (count - 1) if count > 1 else 0
        return variance ** 0.5

    @staticmethod
```

```
    def contar_elementos(arr):
        count = 0
        while True:
            try:
                _ = arr[count]
                count += 1
            except IndexError:
                break
        return count

    @staticmethod
    def calcular_varianza(arr, mean):
        variance_sum = 0
        index = 0

        while True:
            try:
                num = arr[index]
                variance_sum += (num - mean) ** 2
                index += 1
            except IndexError:
                break

        return variance_sum

st_dev_poo = StatsCalculator.st_dev_poo

test_stdev(st_dev_poo)
```

```
..................................................................................
..............................
----------------------------------------------------------------------
Ran 100 tests in 0.178s

OK
```

```
def st_dev_funcional(arr):
    if arr == []:
        raise ValueError("La lista no puede estar vacía.")

    count = reduce(lambda acc, _: acc + 1, arr, 0)
    mean = sum_funcional(arr) / count

    variance_sum = reduce(lambda acc, num: acc + (num - mean) ** 2,
arr, 0)

    variance = variance_sum / (count - 1) if count > 1 else 0
    return variance ** 0.5

test_stdev(st_dev_funcional)
```

```
.....................................................................
.........................
---------------------------------------------------------------------
Ran 100 tests in 0.178s

OK
```

10. Escribe una función para verificar si una cadena es un pangrama, es decir: si contiene todas las letras del alfabeto al menos una vez. Ten en cuenta que las cadenas pueden contener caracteres que no son letras.

```python
def is_pangram_imperativa(s):
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    chars = []
    index = 0

    while True:
        try:
            char = s[index]
            if char.isalpha():
                chars.append(char.lower())
            index += 1
        except IndexError:
            break

    unique_chars = unique_imperativa(chars)

    index = 0
    is_pangram = True

    while True:
        try:
            char = alphabet[index]
            char_found = False
            unique_index = 0

            while True:
                try:
                    if unique_chars[unique_index] == char:
                        char_found = True
                        break
                    unique_index += 1
                except IndexError:
                    break

            if not char_found:
```

```
                is_pangram = False
                break

            index += 1
        except IndexError:
            break

    return is_pangram

# Para testear tu función
test_pangram(is_pangram_imperativa)
```

```
..............................
----------------------------------------------------------------------
Ran 30 tests in 0.048s

OK
```

```
class PangramChecker:
    @staticmethod
    def is_pangram_poo(s):
        alphabet = "abcdefghijklmnopqrstuvwxyz"
        chars = PangramChecker.extraer_letras(s)
        unique_chars = unique_poo(chars)

        return PangramChecker.verificar_pangram(unique_chars,
alphabet)

    @staticmethod
    def extraer_letras(s):
        chars = []
        index = 0

        while True:
            try:
                char = s[index]
                if char.isalpha():
                    chars.append(char.lower())
                index += 1
            except IndexError:
                break

        return chars

    @staticmethod
    def verificar_pangram(unique_chars, alphabet):
        index = 0

        while True:
            try:
                char = alphabet[index]
```

```
                char_found =
PangramChecker.buscar_caracter(unique_chars, char)

                if not char_found:
                    return False

                index += 1
            except IndexError:
                break

        return True

    @staticmethod
    def buscar_caracter(unique_chars, char):
        unique_index = 0

        while True:
            try:
                if unique_chars[unique_index] == char:
                    return True
                unique_index += 1
            except IndexError:
                return False

is_pangram_poo = PangramChecker.is_pangram_poo

test_pangram(is_pangram_poo)

..............................
----------------------------------------------------------------------
Ran 30 tests in 0.053s

OK

def is_pangram_funcional(s):
    alphabet = "abcdefghijklmnopqrstuvwxyz"

    chars = reduce(
        lambda acc, char: acc + [char.lower()] if char.isalpha() else
acc,
        s,
        []
    )

    unique_chars = unique_imperativa(chars)

    contains = lambda char: reduce(
        lambda acc, x: acc or (x == char), unique_chars, False
    )

    is_pangram = reduce(
```

```
        lambda acc, char: acc and contains(char),
        alphabet,
        True
    )

    return is_pangram

test_pangram(is_pangram_funcional)
```

```
.............................
----------------------------------------------------------------------
Ran 30 tests in 0.057s

OK
```

## 11. Escribe una función que reciba una cadena de palabras separadas por comas y devuelva una cadena de palabras separadas por comas ordenadas alfabéticamente.

NOTA: Puedes usar sorted pero no split y definitivamente no join! 

```
def sort_words_imperativa(s):
    words = []
    current_word = ''
    index = 0

    while True:
        try:
            char = s[index]
            if char == ',':
                if current_word:
                    words.append(current_word)
                    current_word = ''
            else:
                current_word += char
            index += 1
        except IndexError:
            break

    if current_word:
        words.append(current_word)

    sorted_words = sorted(words)

    result = ''
    index = 0
    while True:
        try:
```

```
            word = sorted_words[index]
            result += word + ','
            index += 1
        except IndexError:
            break

    return result[:-1] if result else result

# Para testear tu función
test_alpha(sort_words_imperativa)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.168s

OK
```

```
class WordSorter:
    @staticmethod
    def sort_words_poo(s):
        words = WordSorter.extraer_palabras(s)
        sorted_words = sorted(words)
        return WordSorter.formar_resultado(sorted_words)

    @staticmethod
    def extraer_palabras(s):
        words = []
        current_word = ''
        index = 0

        while True:
            try:
                char = s[index]
                if char == ',':
                    if current_word:
                        words.append(current_word)
                        current_word = ''
                else:
                    current_word += char
                index += 1
            except IndexError:
                break

        if current_word:
            words.append(current_word)

        return words

    @staticmethod
    def formar_resultado(sorted_words):
```

```
        result = ''
        index = 0

        while True:
            try:
                word = sorted_words[index]
                result += word + ','
                index += 1
            except IndexError:
                break

        return result[:-1] if result else result

sort_words_poo = WordSorter.sort_words_poo

test_alpha(sort_words_poo)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.144s

OK
```

```
def sort_words_funcional(s):
    words = reduce(
        lambda acc, char: acc[:-1] + [acc[-1] + char] if char != ','
else acc + [''],
        s,
        ['']
    )

    sorted_words = sorted(filter(lambda word: word != '', words))

    result = reduce(lambda acc, word: acc + word + ',', sorted_words,
'')

    return result[:-1] if result else result

test_alpha(sort_words_funcional)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.143s

OK
```

12. Escribe una función para verificar si una contraseña dada es fuerte (al menos 8 caracteres, al menos una minúscula, al menos una mayúscula, al menos un número y al menos un carácter especial). Debería devolver True si es fuerte y False si no lo es. Se permit el uso de regex

```
import re

def is_strong_password_imperativa(password):
    pattern = r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[#@!$%&()^*[\]
{}]).{8,}$'
    return bool(re.match(pattern, password))

# Para testear tu función
test_pass(is_strong_password_imperativa)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.178s

OK
```

```
class PasswordValidator:
    @staticmethod
    def is_strong_password_poo(password):
        pattern = r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[#@!$%&()^*[\]
{}]).{8,}$'
        return bool(re.match(pattern, password))

is_strong_password_poo = PasswordValidator.is_strong_password_poo

test_pass(is_strong_password_poo)
```

```
.....................................................................
.............................
---------------------------------------------------------------------
Ran 100 tests in 0.170s

OK
```

```
pattern = r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[#@!$%&()^*[\]{}]).
{8,}$'

is_strong_password_funcional = lambda password: bool(re.match(pattern,
password))

test_pass(is_strong_password_funcional)
```

```
.....................................................................
.............................
----------------------------------------------------------------------
Ran 100 tests in 0.187s

OK
```