Instituto Superior Tecnológico
GUAYAQUIL

GUÍA GENERAL DE ESTUDIO DE LA ASIGNATURA

FUNDAMENTOS DE PROGRAMACIÓN

DEL INSTITUTO SUPERIOR TECNOLÓGICO
GUAYAQUIL

2023-2024

GUÍA GENERAL DE ESTUDIO DE LA ASIGNATURA

Nombre de la asignatura: Fundamentos de programación
Carrera: Desarrollo de Software
Integrantes: Edgar Andrade, Elena Gallegos, Richard Tigrero
Correo electrónico: eandrade@istg.edu.ec, egallegos@istg.edu.ec, rtigrero@istg.edu.ec

Contenido

Índice de Figuras

INTRODUCCIÓN

Esta Guía se fundamenta de los conocimientos básicos de programación, aplicados a ejercicios reales de las organizaciones y de los diferentes campos de conocimientos, donde se pueda automatizar procesos, con herramientas de programación utilizadas para el desarrollo en la actualidad.

En la Unidad 1, se abordan los conceptos fundamentales de programación enfocados en el lenguaje Python. Se exploran temas como el manejo de tipos de datos, variables y constantes, así como el trabajo con cadenas, lecturas por teclado, operadores lógicos, expresiones anidadas y operadores de asignación.

En la Unidad 2, se abordan conceptos y técnicas de programación que hacen uso de estructuras de control tales como if, while y for. Además, se exploran los tipos de datos estructurados como listas, tuplas, conjuntos y diccionarios. Se incluye la elaboración de menús de opciones que permiten la aplicación práctica de estas estructuras en diferentes contextos.

En la Unidad 3, se establece el entorno de desarrollo a utilizar y se profundiza en la definición de funciones, así como en el manejo del retorno y envío de valores dentro del contexto de la programación.

En la Unidad 4, se profundiza en los conceptos de argumentos y parámetros, así como en el uso de funciones integradas. Además, se exploran técnicas y diseños de algoritmos, donde se ponen en práctica diferentes métodos de resolución y se analiza su funcionamiento en profundidad.

Estas unidades contribuyen al desarrollo del perfil profesional de cada estudiante, ampliando sus conocimientos básicos en el ámbito de la programación.

OBJETIVO GENERAL

Planificar, escribir, desarrollar e implementar algoritmos de programación, a partir del análisis de requerimientos, teniendo en cuenta los criterios de calidad y ética profesional mediante el uso de Python como lenguaje de programación.

OBJETIVO ESPECÍFICO

1. Reconocer escenarios básicos y representarlos en pseudocódigos, desarrollando habilidades en el lenguaje de programación en Python de manera justificada y comprensible.
2. Interpretar algoritmos con estructuras de control, para consolidar las destrezas de desarrollo eficiente y de procedimientos complejos
3. Experimentar instrucciones y sentencias con entorno de desarrollo, utilizando funciones y retornos de valores, enfatizando la toma de decisiones en el proceso de programación.
4. Explorar casos prácticos con técnicas y diseño en algoritmos en Python para dar soluciones en un contexto tecnológico

UNIDAD 1: INTRODUCCIÓN A PYTHON

Objetivo

Reconocer escenarios básicos y representarlos en pseudocódigos, desarrollando habilidades en el lenguaje de programación en Python de manera justificada y comprensible.

Resultado de Aprendizaje

El estudiante comprenderá los conceptos básicos de pseudocodigo para aplicarlos en un lenguaje de programación de alto nivel.

1.1 Análisis Algorítmico

El análisis del algoritmo es el proceso de analizar la capacidad de resolución de problemas del algoritmo en términos del tiempo y el tamaño requeridos (el tamaño de la memoria para el almacenamiento durante la implementación). Sin embargo, la principal preocupación del análisis de algoritmos es el tiempo o rendimiento requerido. El análisis algorítmico consiste en enseñar las bases de la programación y las herramientas básicas con las que se debe aprender a interpretar ejercicios mediante algoritmos y flujogramas.

La necesidad de analizar algoritmos surge como necesidad de eficiencia, es decir elegir un mejor algoritmo para un problema particular, ya que un problema computacional puede resolverse mediante diferentes algoritmos.

Al considerar un algoritmo para un problema específico, podemos comenzar a desarrollar el reconocimiento de patrones de modo que la ayuda de este algoritmo pueda resolver tipos similares de problemas.

Los algoritmos a menudo son bastante diferentes entre sí, aunque el objetivo de estos algoritmos es el mismo. Por ejemplo, sabemos que un conjunto de números se puede ordenar

usando diferentes algoritmos. El número de comparaciones realizadas por un algoritmo puede variar con otros para la misma entrada. Por lo tanto, la complejidad temporal de estos algoritmos puede diferir. Al mismo tiempo, necesitamos calcular el espacio de memoria requerido por cada algoritmo.

En general, realizamos los siguientes tipos de análisis:

- El peor de los casos: el número máximo de pasos dados en cualquier instancia de tamaño N.
- El mejor caso: el número mínimo de pasos dados en cualquier instancia de tamaño N.
- El caso promedio: un número promedio de pasos dados en cualquier instancia de tamaño N.
- El amortizado: una secuencia de operaciones aplicadas a la entrada de tamaño promedio en el tiempo.

1.2 Python como lenguaje de Programación

La historia del lenguaje de programación Python se remonta hacia finales de los 80s y principio de los 90s , su implementación comenzó en diciembre de 1991. Dentro de los lenguajes informáticos Python, pertenece al grupo de los lenguajes de programación y puede ser clasificado como un lenguaje interpretado, de alto nivel, multiplataforma, de tipado dinámico y multiparadigma. A diferencia de la mayoría de los lenguajes de programación, Python nos provee de reglas de estilos, a fin de poder escribir código fuente más legible y de manera estandarizada. Estas reglas de estilo son definidas a través de la Python Enhancement Proposal No 8 (PEP 8).

Un lenguaje informático es un idioma artificial, utilizado por ordenadores, cuyo fin es transmitir información de algo a alguien. Los lenguajes informáticos, pueden clasificarse en:

- Lenguajes de programación (Python, PHP, Pearl, C, etc.).
- Lenguajes de especificación (UML).
- Lenguajes de consulta (SQL).
- Lenguajes de marcas (HTML, XML).
- Lenguajes de transformación (XSLT).
- Protocolos de comunicaciones (HTTP, FTP).

1.3 Tipos de datos

En Python algo importante es que todo se considera un objeto, por lo cual los tipos de datos serían las clases que definen las características y propiedades. Las variables definidas en nuestro código serían las instancias del tipo de dato que le hayamos asignado a cada una de ellas.

Los tipos de datos básicos de Python son los booleanos, los numéricos (enteros, punto flotante y complejos) y las cadenas de caracteres.

Ejemplo:

mi_variable = 15 (Numérico entero)

edad = 043 (Numérico octal)

edad = 0x23 (Numérico hexadecimal)

precio = 7435.28 (Numérico decimal)

verdadero = True / falso = False (Booleano)

mi_cadena = "Hola Mundo!" (Cadena de caracteres)

Python, posee además de los tipos ya vistos, 3 tipos más complejos, que admiten una colección de datos. Estos tipos son:

- Tuplas

- Listas
- Diccionarios

Estos tres tipos, pueden almacenar colecciones de datos de diversos tipos y se diferencian por su sintaxis y por la forma en la cual los datos pueden ser manipulados.

Tuplas

Una tupla es una variable que permite almacenar varios datos inmutables (no pueden ser modificados una vez creados)

Ejemplo:

```
mi_tupla = ('cadena de texto', 15, 2.8, 'otro dato', 25)
```

Listas

Una lista es similar a una tupla con la diferencia fundamental de que permite modificar los datos una vez creados.

Ejemplo:

```
mi_lista = ['cadena de texto', 15, 2.8, 'otro dato', 25]
```

Diccionarios

Los diccionarios permiten utilizar una clave para declarar y acceder a un valor.

Ejemplo:

```
mi_diccionario = {'clave_1': valor_1, 'clave_2': valor_2,'clave_7': valor_7}
```

1.4 Lectura por teclado

La función input() permite obtener texto escrito por teclado. Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Intro.

El uso de la lectura de datos por teclado es uno de los recursos muy útiles a la hora de crear o desarrollar una aplicación en Python, pues mediante esta sentencia el programa deberá guiar o pedir la información necesaria al usuario para el desarrollo de la misma.

Además, esta función puede ser asignada a una variable y sus valores podrán ser transformados al tipo de dato que se necesite para dicha operación.

Ejemplo:

- edad = int(input('Teclear edad: ')) # entrada de entero
- peso = float(input('Teclear peso: ')) # entrada de flotante
- nombre = input('Teclear nombre: ') # entrada de cadena

1.5 Operadores Lógicos

Se utiliza un operador lógico para tomar una decisión basada en múltiples condiciones. Los operadores lógicos utilizados en Python son and, or y not.

- and -> Devuelve True si ambos operadores son True.
- or -> Devuelve True si alguno de los operandos es True.
- not -> Devuelve True si alguno de los operandos False y viceversa.

También se usan para evaluar más de una condición simultáneamente. Los operadores lógicos realizan operaciones lógicas AND, OR y NOT. Generalmente se usan junto con los operadores relacionales para poder crear diferentes caminos dentro de nuestro código según las condiciones se vayan cumpliendo.

Ejemplo:

Operador	**Ejemplo**	**Resultado**
and (y)	5 == 7 and 7 < 12	0 y 0 Falso
or (o)	12 == 12 or 15 < 7	1 o 0 Verdadero

1.6 Expresiones Anidadas

Los condicionales, permiten escribir código en su interior y en realidad, nada impide incluso al interior de un condicional, poner otros (u otros). A eso se le llama condiciones anidados, pues una estructura condicional dentro de otra. De hecho, puedes anidar cuantos condicionales requieras, aunque no se recomienda más de dos o tres niveles.

Se pueden solucionar empleando las reglas de precedencia:

- Primero los paréntesis que indican prioridad.
- Segundo, las expresiones aritméticas por sus propias reglas.
- Tercero, las expresiones relacionales.
- Cuarto, las expresiones lógicas.

Dentro de un condicional, puedes poner cualquier instrucción válida y eso incluye a cualquier tipo de condicional que necesites y el funcionamiento, sigue siendo el mismo. De hecho, Python tiene una instrucción llamada elif() que permite simplificar un if() que se encuentra al interior de un else en Python.

Ejemplo:

```
password = input("Ingrese la contraseña: ")
if (len(password) >= 8):
        print('Tu contraseña es suficientemente larga.')
        if(password == 'miClaveSegura'):
                print("Además es la contraseña correcta.")
        else:
                print("Pero es incorrecta.")
else:
```

```
            print('Tu contraseña es muy corta e insegura.')
    if (password != 'miClaveSegura'):
        print("Además, es incorrecta (por supuesto).")
```

1.7 Operadores de Asignación

Los operadores de asignación nos permiten realizar una operación y almacenar su resultado en la variable inicial. Existe en Python todo un grupo de operadores los cuales le permiten básicamente asignar un valor a una variable, usando el operador "=". Con estos operadores pueden aplicar la técnica denominada asignación aumentada.

Operador igual (=)

El operador igual prácticamente no necesita explicación, simplemente asigna a la variable de la izquierda el contenido que le ponemos a la derecha.

Operador más igual (+=)

Como podemos ver, todos los operadores de asignación no son más que atajos para escribir otros operadores de manera más corta, y asignar su resultado a la variable inicial.

El operador += en x+=1 es equivalente a x=x+1.

Ejemplo:

```
x=5         # Ejemplo de como incrementar
x+=1        # en una unidad x
print(x)   # 6
```

Operador menos igual (-=)

Como podemos ver, todos los operadores de asignación no son más que atajos para escribir otros operadores de manera más corta, y asignar su resultado a la variable inicial.

El operador -= en x-=1 es equivalente a x=x-1.

Ejemplo:

```
x=5          # Ejemplo de cómo incrementar
x-=1         # en una unidad x
print(x)   # 4
```

UNIDAD 2: ESTRUCTURAS DE CONTROL Y COLECICONES

Objetivo

Interpretar algoritmos con estructuras de control, para consolidar las destrezas de desarrollo eficiente y de procedimientos complejos.

Resultado de Aprendizaje

El estudiante será capaz de programar diferentes estructuras de control para ejecutar decisiones automatizadas en base a parámetros definidos

2.1 Estructuras de control

Las estructuras de control son elementos fundamentales en la programación que permiten controlar el flujo de ejecución de un programa. Estas estructuras permiten tomar decisiones y repetir ciertas acciones según condiciones específicas. Las principales estructuras de control son:

Estructuras de control condicional:

If (si): Permite ejecutar un bloque de código si una condición es verdadera.

Else (sino): Se utiliza junto con "if" para ejecutar un bloque de código cuando la condición es falsa.

Elif (sino si): Permite evaluar múltiples condiciones en orden.

Estructuras de control de bucles (ciclos):

While (mientras): Permite ejecutar un bloque de código mientras una condición sea verdadera.

For (para): Se utiliza para iterar sobre una secuencia (como una lista o rango) y ejecutar un bloque de código para cada elemento.

Estructuras de control de salto:

Break (romper): Se utiliza para salir de un bucle antes de que la condición de este se cumpla.

python

Continue (continuar): Permite saltar a la siguiente iteración de un bucle sin ejecutar el resto del código dentro del bucle.

Estas estructuras proporcionan flexibilidad y control en la programación, permitiendo que los programas se adapten dinámicamente a diferentes situaciones. La elección de la estructura adecuada depende de la lógica específica que se desee implementar en el programa.

2.2 Sentencia IF

De no ser por las estructuras de control, el código en cualquier lenguaje de programación sería ejecutado secuencialmente hasta terminar. Un código, no deja de ser un conjunto de instrucciones que son ejecutadas unas tras otra. Gracias a las estructuras de control, podemos cambiar el flujo de ejecución de un programa, haciendo que ciertos bloques de código se ejecuten si y sólo si se dan unas condiciones particulares.

Es muy importante tener en cuenta que la sentencia if debe ir terminada por: y el bloque de código a ejecutar debe estar identado. Si usas algún editor de código, seguramente la identación se producirá automáticamente al presionar enter.

Ejemplo:

```
a = 4
b = 2
if b != 0:
    print(a/b)
```

En este ejemplo podemos ver cómo se puede usar un if en Python. Con el **operador != se** comprueba que el número b sea distinto de cero, y si lo es, se ejecuta el código que está indentado. Por lo tanto, un if tiene dos partes:

- La condición que se tiene que cumplir para que el bloque de código se ejecute, en nuestro caso b!=0.
- El bloque de código que se ejecutará si se cumple la condición anterior.

2.3 Sentencia While

Un bucle while permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras la condición tenga el valor True). Los bucles while tienen la particularidad de no tener un fin propio dentro del bucle a menos que una condición interna dentro de los procesos que repite el bucle hagan que la función de la función. En el siguiente código se muestra la estructura del bucle while y se explica su funcionamiento.

```
x=0
while (x<=10):
   print(x,' Hola mundo')
   x+=1
```

Esta porción de código está declarando una variable a la que se le ha llamado x con un valor inicial cero (0) para luego indicar en el bucle while que, mientras el valor que x tiene sea menor o igual que cero el proceso interno se seguirá ejecutando. Una vez ingresado al bucle, se está realizando la impresión del número que tiene x concatenado con la cadena de caracteres 'Hola mundo'. Este proceso se repetirá once veces ya que x va a ir incrementando su valor en una por cada iteración que se vaya dando. A continuación, se muestra la impresión que se da en la consola cuando se ejecuta este comando.

```
0  Hola mundo
1  Hola mundo
2  Hola mundo
3  Hola mundo
4  Hola mundo
5  Hola mundo
6  Hola mundo
7  Hola mundo
8  Hola mundo
9  Hola mundo
10  Hola mundo
```

2.4 Sentencia FOR

A diferencia del bucle while tiene como finalidad la repetición de procesos bueno en un rango limitado de veces. Es decir, dentro del bucle se colocan 1, 2 o 3 argumentos los cuales me indicarán donde inicia, donde finaliza, y de cuánto en cuánto van a ser los incrementos que se

van a ir realizando dentro de nuestro bucle. Explicado lo anterior cabe recalcar que al menos podemos crear 3 tipos de bucles.

El primer bucle únicamente tiene un argumento, en el cual indicará en qué número finaliza el rango de nuestro bucle por lo tanto se infiere que empieza en cero y que el incremento se va a ir realizando de 1 en 1.

```
for x in range (10):
   print(x, 'Hola mundo')
```

El segundo bucle posee dos argumentos, primer momento indica en qué número inicia nuestro bucle y el segundo argumento indica en qué número finaliza nuestro bucle, al igual que el anterior el incremento se irá realizando de 1 en 1 dado que no se está indicando lo contrario.

```
for x in range (0,10):
   print(x, 'Hola mundo')
```

El tercer bucle posee 3 argumentos, el primer argumento indicará en quien número inicia nuestro bucle, el segundo argumento indicará en qué número finaliza nuestro bucle, y el último argumento indicará de cuánto en cuánto se va incrementando los saltos de nuestra variable dependiente en el bucle.

```
for x in range (0,10,2):
   print(x, 'Hola mundo')
```

2.5 Listas

Las listas en Python son colecciones ordenadas de datos las cuales se les puede ingresar todo tipo de datos u otras sub colecciones de datos, es decir en las listas podemos ingresar datos de tipo entero, datos de tipo decimal, datos de tipo carácter, podemos ingresar listas dentro de otras listas utilizando los corchetes, podemos ingresar tuplas diccionarios y conjuntos dentro de una lista principal. todos estos datos que se encuentran dentro de las listas pueden ser mostrados en pantalla o en consola utilizando la posición en la que se encuentra el elemento. Para poder imprimir o mostrar un dato en pantalla de una lista es necesario conocer la posición

teniendo en cuenta que las posiciones en las listas se las empieza a contar desde cero en adelante de izquierda a derecha. En caso de que desee imprimir los elementos de derecha a izquierda la contabilización de las posiciones se hace con los números negativos y empezando desde menos 1 para poder imprimir el último elemento de la lista de esta manera se efectiviza las impresiones de la lista de izquierda a derecha y de derecha a izquierda acorde a la necesidad del usuario.

Ejemplo

```
x=[2,'a', [10], ('tp', 25), {15,10, 'Juan'}]
```

En el ejemplo mostrado a continuación la lista presenta todo tipo de datos sub listas conjuntos de tuplas ingresados dentro de la misma

2.6 Tuplas

Las tuplas son un tipo de colección de datos particular de Python que como característica principal es inmutable, lo que quiere decir que una vez creada una tupla como una x cantidad de elementos, estos elementos son inmodificables lo que quiere decir que no puedo agregar quitar eliminar o cambiar estos elementos dentro de la tupla. otra particularidad que poseen las tuplas en Python es que son ordenadas es decir poseen indexación lo que corresponde a poder extraer o imprimir los elementos dentro de la tupla acorde a la posición en la que se encuentre, y al igual que en las listas, puedo imprimir los elementos de izquierda a derecha y de derecha a izquierda indicando la posición en la que se encuentre. si se los imprime de izquierda a derecha basta con indicar desde la posición cero hasta la posición n dónde está el elemento para poderlo mostrar en pantalla. En cambio, si voy a imprimir un elemento visualizándolo de derecha a izquierda debo tener en cuenta que la posición inicial de derecha a izquierda empieza en menos 1 y de esta manera puedo imprimir el último elemento de la lista hacia atrás. Además, puedo imprimir elementos por rango de posiciones colocando entre corchetes el rango de la posición en la cual se encuentra el elemento. Sí colocó un rango inicial y un rango final separado por dos puntos, se imprimirán los elementos que se encuentren desde el rango inicial

hasta una posición antes del rango final. En el caso de colocar únicamente el rango inicial y los dos puntos, si imprimirán todos los elementos a partir del rango final hasta el final de la tupla, por el contrario, sí primero colocó dos puntos y luego el rango, se imprimirán todos los elementos desde el inicio hasta el rango final que yo ubique.

```
x=(2,'pepe',5,'juan')
print(x)
print(x[-1])
print(x[0:2])
print(x[1:])
print(x[:3])
```

2.7 Conjuntos

Los conjuntos Son una colección de datos son una colección de datos en los cuales se pueden guardar cualquier tipo de datos listas tuplas o subconjuntos de forma desordenada, caracterizados principalmente por utilizar ya ves y el carácter coma de separación. Los conjuntos tienen la particularidad de no receptar elementos repetidos al igual que en las matemáticas. Esto nos ayuda a guardar información dentro de un conjunto con la seguridad de que no va a haber un elemento repetido en el mismo y así evitar la duplicidad de información. Los conjuntos a diferencia de las tuplas y de las listas, no posee indexación es decir los datos que se encuentran dentro de un conjunto no son ordenados por lo tanto para imprimir un elemento del conjunto tengo que indicarle a la consola de comandos el nombre del elemento que voy a imprimir ya que no lo puedes traer por la posición en la que se ubica, de igual manera si se imprime un conjunto en su totalidad se podrá notar que al momento de la creación del conjunto o de la inserción de elementos dentro de un conjunto tienen u ocupan una posición temporal, aunque luego cuando son mostrados en pantalla el compilador cambia estas posiciones, ejecutándose así la característica del conjunto que indica que sus elementos se encuentran de forma desordenada. Dentro de los conjuntos se pueden agregar eliminar quitar o modificar los elementos que posea además de que al igual que en las matemáticas también se

pueden realizar operaciones entre conjuntos como suma entre conjuntos diferencia entre conjuntos Unión entre conjuntos intersección entre conjuntos etcétera.

Ejemplo de conjuntos

```
x={2,5,'Juan', (9,10)}
print(x)
```

2.8 Diccionarios

Los diccionarios son poderosas estructuras de datos en Python que almacenan datos como pares de claves, siendo está representada en la siguiente forma: Clave-Valor. La comprensión de diccionarios (o dict comprehension) puede ser muy útil en crear nuevos diccionarios basados en diccionarios existentes e iterables.

Los diccionarios en Python nos permiten almacenar una serie de mapeos entre dos conjuntos de elementos, llamados keys and values (Claves y Valores).

Todos los elementos en el diccionario se encuentran encerrados en un par de corchetes {}.

Cada elemento en un diccionario contiene una clave y un valor - es decir un par de clave-valor.

Cada par de clave-valor es denominado como elemento (ítem).

La ventaja de esto es que puedes acceder a todos los valores almacenados usando simplemente las claves.

He aquí un ejemplo de la estructura pertinente a un diccionario:

```
mi_diccionario = {"key1":'value1',"key2":'value2',"key3":'value3'}
```

El diccionario mi_diccionario contiene tres pares de clave-valor, es decir, cuatro elementos.

"key1" hasta "key3" son las tres claves.

Puedes usar mi_diccionario["key1"] para acceder a <value1>, y mi_diccionario["key2"] para acceder a <value2>, y así sucesivamente.

UNIDAD 3: FUNCIONES EN PYTHON

Objetivo

Experimentar instrucciones y sentencias con entorno de desarrollo, utilizando funciones y retornos de valores, enfatizando la toma de decisiones en el proceso de programación.

Resultado de Aprendizaje

El estudiante podrá hacer uso de funciones y métodos en la construcción de un programa mediante el diseño de una solución utilizando instrucciones avanzadas

3.1 Entorno de Desarrollo

Un entorno de desarrollo, también conocido como IDE (del inglés Integrated Development Environment), es un conjunto de herramientas integradas diseñadas para facilitar el desarrollo de software. Estos entornos proporcionan un conjunto de características que incluyen editores de código, depuradores, compiladores/intérpretes, herramientas de gestión de proyectos, entre otros, todo integrado en una sola interfaz de usuario.

Para trabajar con Python, hay varios entornos de desarrollo populares disponibles. Aquí hay algunos de ellos:

PyCharm: Como se mencionó anteriormente, PyCharm es uno de los IDE más populares para el desarrollo de Python. Es desarrollado por JetBrains y ofrece una amplia gama de características para facilitar el desarrollo de aplicaciones Python.

Visual Studio Code (VS Code): Aunque no es específico para Python, VS Code es un editor de código muy popular y altamente personalizable que es ampliamente utilizado por desarrolladores de Python. Viene con extensiones que permiten la integración con Python, incluyendo soporte para depuración, autocompletado y gestión de entornos virtuales.

Spyder: Spyder es un entorno de desarrollo científico diseñado específicamente para Python y es especialmente popular entre los científicos de datos. Viene preinstalado con muchas bibliotecas científicas populares y ofrece herramientas avanzadas para análisis de datos y visualización.

Jupyter Notebook / JupyterLab: Jupyter es una plataforma interactiva que permite la creación y el intercambio de documentos que contienen código, visualizaciones y texto narrativo. Es muy utilizado en el campo de la ciencia de datos y el análisis exploratorio de datos.

Atom: Similar a VS Code, Atom es un editor de texto altamente personalizable que puede ser extendido para admitir el desarrollo de Python con complementos. Aunque no es un IDE específico de Python, es una opción popular entre los desarrolladores de Python debido a su flexibilidad y extensibilidad.

IDLE: IDLE es el IDE incluido con la instalación estándar de Python. Es simple y fácil de usar, lo que lo convierte en una opción adecuada para principiantes que están aprendiendo Python.

Estos son solo algunos de los entornos de desarrollo disponibles para trabajar con Python. La elección de un IDE depende de las preferencias personales del desarrollador, así como de los requisitos específicos del proyecto.

3.2 Funciones

En los lenguajes de programación como python, las funciones son bloques de código reutilizables que realizan una tarea específica. Estas funciones pueden tomar cero o más argumentos como entrada, realizar ciertas operaciones y, opcionalmente, devolver un resultado.

3.3 Definición de funciones

En general las funciones en los lenguajes de programación son conjuntos de instrucciones que se escriben separadamente y que realizan alguna tarea especificada. Las funciones pueden usarse directamente en la ventana interactiva, o desde programas, o integrarlas en un módulo especial a manera de librería para organizar el desarrollo de un proyecto de programación.

Esta estrategia de dividir la resolución de un problema en módulos y funciones es parte de la metodología denominada Programación Modular que es importante para resolver problemas grandes o complejos. La Programación Modular facilita el diseño, construcción, prueba e integración de los componentes de un programa. El mecanismo usual para transmitir datos a las funciones es mediante una lista de variables que se denominan parámetros. Las funciones normalmente son diseñadas para entregar resultados.

Hay dos tipos de funciones:

- funciones incorporadas, que son las funciones que vienen con python, como son las siguientes: print(), input(), int(), float(), type(), etc.
- funciones definidas por nosotras, que son las que creamos, para luego usarlas de donde sean invocadas.

Declaración de una función:

```
def nombre(parámetros):
    instrucciones
```

nombre: Es la identificación de la función. Puede contener una palabra o **n** palabras seguidas de un conector, ejemplo “_”. Los nombres en el lenguaje Python admiten tildes y la letra ñ

parámetros: Son variables que reciben los datos que entran a la función.

instrucciones: Se incluyen en la función para producir resultados

Las instrucciones incluidas en la función deben estar encolumnadas como en las otras estructuras de control.

Una función puede tener un número arbitrario de parámetros separados por comas.

Aquí un ejemplo básico de una definición de función con el propósito de imprimir Hola cada vez que esta sea llamada:

```
def saludo():
   print("Hola")
```

Otro ejemplo de una definición de una función con una variable parámetro, el mismo que es impreso cada vez que se lo invoque.

```
def saludo _dos(saludo):
        print(saludo)
```

3.4 Retorno de valores

Una función está diseñada para entregar un resultado, toda función sea esta de cualquier tipo retorna uno o más resultados por medio de la instrucción return, palabra clave que retorna el resultado o los resultados donde se la llame o invoque.

```
def nombre(parámetro1, parámetro2,...):
        instrucciones
        return variable
```

Donde variable: es la identificación de la variable que contiene el resultado.

Una función puede entregar más de un resultado en una lista: **return [variable,variable,...]**

o separado por " , " return variable,variable,...

¿Procedimientos o funciones?

En algunos lenguajes de programación las funciones que no retornan ningún valor en su llamada son llamadas procedimientos. Todas los procedimientos en python son funciones.

Ejemplo: Escriba una función matemática:

```
y = f(x)= 2x°2  + 1
 Definición de la función:
 def f(x):
     y=2*x**2 + 1
      return y
```

Si una función tiene una forma simple, puede escribirse en forma abreviada en una línea

```
>>> def f(x): return 2*x**2+1
```

Las funciones en python pueden retornar cualquier tipo de dato por medio de la instrucción return. Una función puede retornar varios tipos de datos en ella misma.

```
def many_types(x):
    if x < 0:
        return "Hello!"
    else:
        return 0
```

Ejemplo: Definición de una función llamada suma

```
def suma(parametro1,parametro2):
        resultado = parametro1 + parametro2
        return resultado
```

3.5 Envió de valores

En una función los envíos de valores se realizan en las invocaciones a las funciones en el programa o de donde sea solicitada.

Un módulo puede incluir varias funciones. Estos módulos pueden considerarse librerías.

Los beneficios que ofrecen las funciones en un lenguaje de programación son bien conocidos. El beneficio principal, sin duda alguna, es la reutilización de código, o sea, la idea de tener en un fragmento de código una funcionalidad pueda usarse,ejecutarse una y otra vez sin la necesidad de codificar cada vez que se utilice.

Las funciones en un sentido positivo son contenedores de código que deben cumplir un determinado objetivo. Cuando se solicita la ejecución de una función se efectúa un llamado a función y realizan una sentencia donde aparezca el nombre de la función seguido de paréntesis donde se indiquen los parámetros que requiere, en el caso de que requiera alguno.

Ejemplo1: Defina un módulo con el nombre funciones para almacenar las funcione

Ilustración 1. *Ejercicio de módulo*

$f(x)=2x^2+1$
$g(x)=3x^3+5$

```
def f(x):
    y=2*x**2 + 1
    return y
def g(x):
    y=3*x**3 + 5
    return y
```

Fuente: Luis Rodriguez Ojeada - Libro digital. Versión 2.5 2016, Python Programación

En la Figura 1 vemos que en el recuadro pertenece al archivo **fuciones**.py donde se encuentran las funciones que se han definido.

Para usar las funciones en el archivo de Python **main**.py debe importar el módulo

```
from funciones import*
resultF =  f(3) #llamamos a la funcion “f” definida en el módulo
print(result)
```

Salida:
19

```
resultG= g(4) #llamamos a la función “g” definida en el módulo
print(resultG)
```

Salida:
197

Nótese que en las llamadas a las funciones se envían valores enteros, esto es porque la función está definida con una variable de parámetro que por la naturaleza del ejercicio su tipo de dato debe ser numérico.

Los programas que requieran usar las funciones pero que se escriben separadamente de las definiciones de las funciones, deben importar el módulo

#Programa que usa el módulo funciones

```
from funciones import*
for i in range(5):
    y=f(i)
    print(i,y)
```

Prueba del programa

```
>>>
0
1
1
3
2
9
3
1
9
4
3
3
 >>>
```

Note que la llamada a la función está dentro de un ciclo, lo que significa que la función va hacer invocada la cantidad de veces que se ejecute el mismo. En este caso note que la función "f" es llamada 5 veces enviando el valor de **i** en cada iteración, siendo el valor de **i** diferente en cada ciclo.

Ejemplo2: Defina un módulo (librería) con el nombre geometría que contenga funciones con algunas fórmulas de la geometría básica para calcular área y volumen. Incluya fórmulas para el círculo, sector de círculo, segmento de círculo, esfera, cilindro y cono.

Ilustración 2. Ejercicio módulos

```
from math import*
def circulo(r):
    s=pi*r**2
    return s

def sector(r,a):
    s=pi*r**2*a
    return s

def segmento(r,a):
    s=r**2*(pi*a-0.5*sin(a))
    return s

def esfera(r):
    s=4*pi*r**2
    v=4*pi*r**3/3
    return [s,v]

def cilindro(r,h):
    s=2*pi*r*(r+h)
    v=pi*r**2*h
    return [s,v]

def cono(r,h):
    g=sqrt(h**2+r**2)
    s=pi*r*g+pi*r**2
    v=pi*r**2*h/3
    return [s,v]
```

Fuente. Luis Rodríguez Ojeada - Libro digital. Versión 2.5 2016, Python Programación

Para usar las funciones desarrolladas, debe importar el módulo en la ventana interactiva o en algún programa.

En el siguiente fragmento de código se va a usar el módulo geometría con las funciones

```
>>> from geometria import*
 >>> circulo(2)
12.566370614359172
 >>> s=circulo(2)
>>> print(s)
12.566370614359172
>>> cilindro(4,5)
 (226.1946710584651, 251.32741228718345)
>>> [s,v]=cilindro(4,5)
>>> print(s)
 226.1946710584651
>>> print(v)
251.32741228718345
```

Ejemplo3: Escriba una función que reciba un número y retorne el 50% de este. El resultado que entrega la función será un valor lógico según corresponda.

Junto a la función escriba un programa que liste los porcentajes de los números en un rango especificado

Ilustración 3. Ejercicio por ciento

```
#FUNCIONES
#FUNCIÓN QUE CALCULA EL 50% DE UN VALOR
def fun_porciento(n):
    resultado = n*0.5
    return resultado

#PROGRAMA QUE LISTA % DE UN RANGO
a=int(input('Desde: '))
b=int(input('Hasta: '))
for n in range(a,b+1):
    porciento = fun_porciento(n)
    print(f'El 50 porciento de {n} es : ',porciento)
```

Note que en este ejercicio la función está dentro del programa principal main.py, esto quiere decir que también lo podríamos hacer, sin embargo, es recomendable realizarlo en otro archivo.

Función booleana

Esta función son las que retornan un True o un False cuando el valor es verdadero o falso respectivamente. Son funciones que se utilizan para validar instrucciones.

Por ejemplo:

```
def esReal(n):
    if n >=0 or n<0:
        return True
    else:
        return false
```

Note que para cada valor de n mayor o igual o menor a 0 es real, es decir todos los números reales, si se cumple esta condición la devolución es True caso contrario será False.

Ejercicio 4: Solicitar al usuario que ingrese su dirección de email. Imprimir un mensaje indicando si la dirección es válida o no, valiéndose de una función para decidirlo. Una dirección se considerará válida si contiene el símbolo "@".

Ilustración 4. Solución ejercicio 4

Solución:

```
def validar(email):
    caracterBuscado="@"
    emailValido=False
    for c in email:
        if c==caracterBuscado:
            return True
    return False


direccion=input("Tu email: ")
if validar(direccion):
    print("Dirección válida")
else:
    print("Dirección inválida")
```

Fuente. Programación desde cero: http://patriciaemiguel.com/ejercicios/python/2019/03/10/ejercicios-funciones-python.html

UNIDAD 4: TÉCNICAS Y DISEÑO DE ALGORITMOS EN PYTHON

Objetivo

Explorar casos prácticos con técnicas y diseño en algoritmos en Python para dar soluciones en un contexto tecnológico

Resultado de Aprendizaje

En estudiante estará capacitado para analizar diferentes casos con programación avanzada

4.1 Argumentos y parámetros

De la forma más concreta vamos a decir que los parámetros son esas variables que se definen en las funciones las cuales pueden ser del nombre que desee sin dejar de representar lo que se está programando. Con respecto a los argumentos, los argumentos son esos valores que se envían a la función cuando ésta es invocada, por ejemplo:

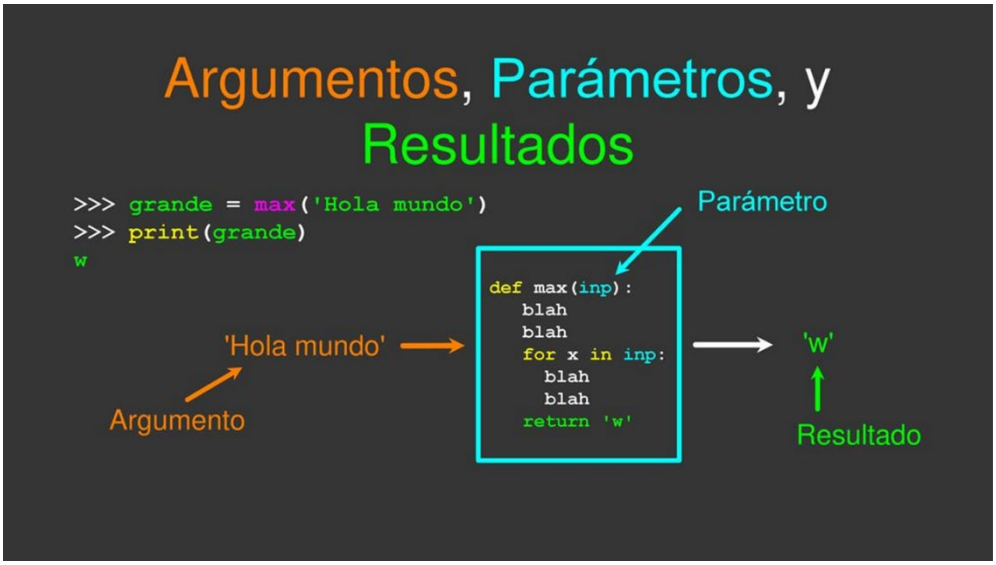
Definición de función:

```
def nombreFun(parámetro1, parámetro2):
        return result
    - parámetro1, parámetro2 #son las variables de parámetro
```

Invocación a la función:

```
var1 = 4
var2 = 5
salida = nombreFun(var1,var2)
    - var1,var2 # son los argumentos o valores enviados a la función
```

Ilustración 5. Diferencia entre argumentos y parámetros



Fuente: Python para todos: https://slideplayer.es/amp/17516386/

La mayoría de las funciones necesitan ser definidas con parámetros, y cada lenguaje tiene sus propias especificaciones de cómo definir estos parámetros en funciones. Python es flexible y provee tres opciones para definir estos parámetros y argumentos en las funciones.

Parámetro por posición

Cuando enviamos argumentos a una función, estos se reciben por orden en los parámetros definidos. Se dice por tanto que son argumentos por posición

Ilustración 6. Parámetros por posición

```
def resta(a, b):
    return a - b

resta(30, 10)  # argumento 30 => posición 0 => parámetro a
               # argumento 10 => posición 1 => parámetro b
```

```
20
```

Fuente: https://docs.hektorprofe.net/python/programacion-de-funciones/argumentos-y-parametros/

Pasando argumentos por nombre de parámetros

Sin embargo, es posible evadir el orden de los parámetros si indicamos durante la llamada que valor tiene cada parámetro a partir de su nombre

Ilustración 7. Parámetros por nombre

```
resta(b=30, a=10)
```

```
-20
```

Fuente: https://docs.hektorprofe.net/python/programacion-de-funciones/argumentos-y-parametros/

En donde a y b son parámetros de la función resta

Nosotros podemos definir la función con arbitrario número de parámetros con el símbolo de *:

```
def func(*param):  #param será una tupla con los valores pasados desde la invocación
    for i in param:
        print(i)

func(1, 2, 3)  #llamada con 3 argumentos

# Salida:
#1
# 2
# 3
```

Llamada sin argumentos

Al llamar una función que tiene definidos unos parámetros, si no pasamos los argumentos correctamente provocará un error:

Ilustración 8. Llamada sin argumentos

```
resta()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-78c8f433960e> in <module>()
----> 1 resta()

TypeError: resta() missing 2 required positional arguments: 'a' and 'b'
```

Fuente: https://docs.hektorprofe.net/python/programacion-de-funciones/argumentos-y-parametros/

Paso por valor y referencia

Dependiendo del tipo de dato que enviemos a la función, podemos diferenciar dos comportamientos:

- **Paso por valor**: Se crea una copia local de la variable dentro de la función.
- **Paso por referencia**: Se maneja directamente la variable, los cambios realizados dentro de la función le afectarán también fuera.

Tradicionalmente

- **Los tipos simples se pasan por valor**: Enteros, flotantes, cadenas, lógicos...
- **Los tipos compuestos se pasan por referencia**: Listas, diccionarios, conjuntos...

Ejemplo de paso por valor:

Como a sabemos los números se pasan por valor y crean una copia dentro de la función, por eso no les afecta externamente lo que hagamos con ellos:

Ilustración 9. Paso por valor

```
def doblar_valor(numero):
    numero *= 2

n = 10
doblar_valor(n)
print(n)
```

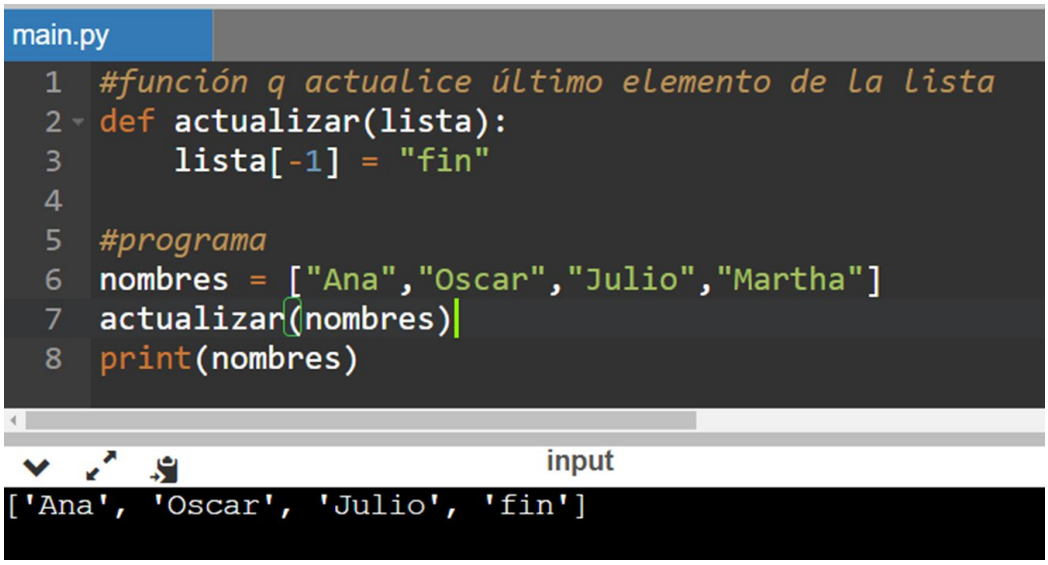
```
10
```

Fuente: https://docs.hektorprofe.net/python/programacion-de-funciones/argumentos-y-parametros/

Ejemplo de paso por referencia

Sin embargo, las listas u otras colecciones, al ser tipos compuestos se pasan por referencia, y si las modificamos dentro de la función estaremos modificándose también fuera:

Ilustración 10. Paso por referencia



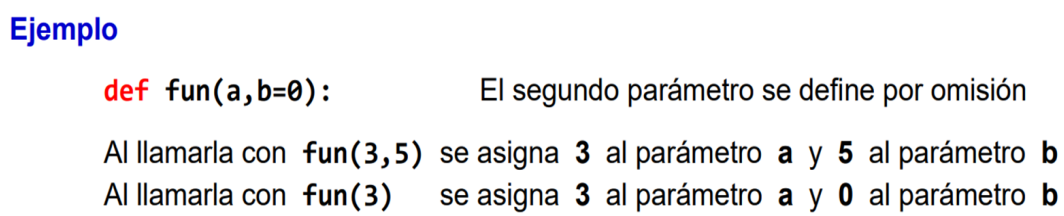
Nótese que lo que le suceda a la lista interna en la función le sucederá a la lista externa en el programa

Parámetros por omisión

Es posible asignar valores a los parámetros para el caso que no vengan con algún valor al ser llamada la función.

Esto significa que una función puede ser llamada con menos parámetros que los que se especifican en la definición.

Ilustración 11. Parámetros por omisión



Fuente: Luis Rodríguez Ojeada - Libro digital. Versión 2.5 2016, Python Programación

4.2 Funciones integradas

Las funciones integradas en Python son aquellas que están disponibles directamente en el lenguaje sin necesidad de importar ningún módulo adicional. Son parte del núcleo del lenguaje

y proporcionan funcionalidades básicas y útiles. Aquí hay algunas funciones integradas comunes en Python:

print(): Esta función se utiliza para imprimir mensajes en la salida estándar, como la consola. Es muy útil para depurar y mostrar información durante la ejecución del programa.

```
print("Hola, mundo!")
```

len(): Devuelve la longitud de un objeto, como una cadena, lista, tupla, diccionario o conjunto.

```
cadena = "Hola"

print(len(cadena))  # Output: 4
```

input(): Lee una entrada del usuario desde la consola y devuelve una cadena de caracteres.

```
nombre = input("Ingrese su nombre: ")

print("Hola,", nombre)
```

range(): Genera una secuencia de números enteros dentro de un rango especificado.

```
for i in range(5):
    print(i)  # Output: 0 1 2 3 4
```

type(): Devuelve el tipo de datos de un objeto.

```
numero = 42

print(type(numero))  # Output: <class 'int'>
```

sum(): Devuelve la suma de los elementos de un iterable (lista, tupla, etc.)

```
numeros = [1, 2, 3, 4, 5]
```

print(sum(numeros)) # Output: 15

max() y min(): Devuelven el valor máximo y mínimo de un iterable, respectivamente.

numeros = [1, 2, 3, 4, 5]

print(max(numeros)) # Output: 5

print(min(numeros)) # Output: 1

abs(): Devuelve el valor absoluto de un número.

numero = -10

print(abs(numero)) # Output: 10

Estas son solo algunas de las funciones integradas en Python. Existen muchas otras funciones útiles que pueden facilitar el desarrollo de programas en Python. Puedes encontrar una lista completa en la documentación oficial de Python.

4.3 Casos prácticos (Algoritmos de ordenamiento)

Dentro de los casos prácticos de funciones que podemos tener a más de los que se ha explicado en esta unidad, se listan los siguientes:

Ilustración 12. Función de cálculo

Ejemplo.

```
def f(x):
    y=2*x**2 + 1
    return y

#Programa que usa la función f
for i in range(5):
    y=f(i)
    print(i,y)
```

Fuente. Luis Rodríguez Ojeada - Libro digital. Versión 2.5 2016, Python Programación

Funciones condicionadas

Ejemplo. Escriba una función que reciba un número y determine si es un número primo. El resultado que entrega la función será un valor lógico según corresponda.

Junto a la función escriba un programa que liste los números primos existentes en un rango especificado

Solución

Si el programa se escribe junto con la función, primero debe escribirse la función y después el programa. En este caso, el programa no necesita importar la función que lo antecede.

Variables

n: número para probar si es primo

c: conteo de números divisores en la función

a,b: rango de búsqueda de número primos

La función entrega un resultado lógico: True o False

Ilustración 13. Función primo

```
#función primo
def primo(n):
    c=0
    for i in range(1,n+1):
         if n%i==0:
             c=c+1
    if c>2:
        return False
    else:
        return True

#Programa que lista primos en un rango
a=int(input('Desde: '))
b=int(input('Hasta: '))
for n in range(a,b+1):
      if primo(n):
            print('Número primo: ',n)
```

Fuente: Luis Rodríguez Ojeada - Libro digital. Versión 2.5 2016, Python Programación

Prueba del programa

>>>
Desde: 20
Hasta: 40

Número primo: 23
Número primo: 29
Número primo: 31
Número primo: 3

Nota: Las instrucciones de prueba se las ha escrito junto a la función. Esta es una buena práctica para desarrollar funciones. Después se la puede almacenar separadamente para que pueda ser importada desde cualquier programa o desde la ventana interactiva.

Funciones con tipo de datos estructurados: Función que recibe un diccionario y retorna otro diccionario con los precios finales de una lista de productos.

Solución

Si el programa se escribe junto con la función, primero debe escribirse la función y después el programa. En este caso, el programa si necesita importar la función que lo antecede.

Variables

prod: base de productos

precio_final: variable que almacena el precio final del producto

dic_pre: diccionario resultante

La función entrega una variable de tipo diccionario

#diccionario de productos con clave str y valores en lista de dos elementos

Ilustración 14. Función precio producto

```
def precio_pro(prod):
    dic_pre = {"codpro":0} #inicializamos el diccionario a retornar
    for key in prod:
        precio_final = prod[key][0] - prod[key][1]
        dic_pre[key]=precio_final
    return dic_pre #retorna un diccionario con clave str y valor int
```

Archivo funciones.py

```
from funciones import*
```

Ilustración 15. Programa precio producto

```
#programa
producto2 = {'001':[6,2],'002':[10,5],'003':[20,5]}
dic_ = precio_pro(producto2)
for key in dic_:
    print(f'Producto: {key}, precio: {dic_[key]}')
```

Prueba del programa:

Producto: 001, precio: 4
Producto: 002, precio: 5
Producto: 003, precio: 15

Funciones con más de un retorno: Función separar lista, esta función recibe una lista de números enteros y separa en una lista de pares e impares.

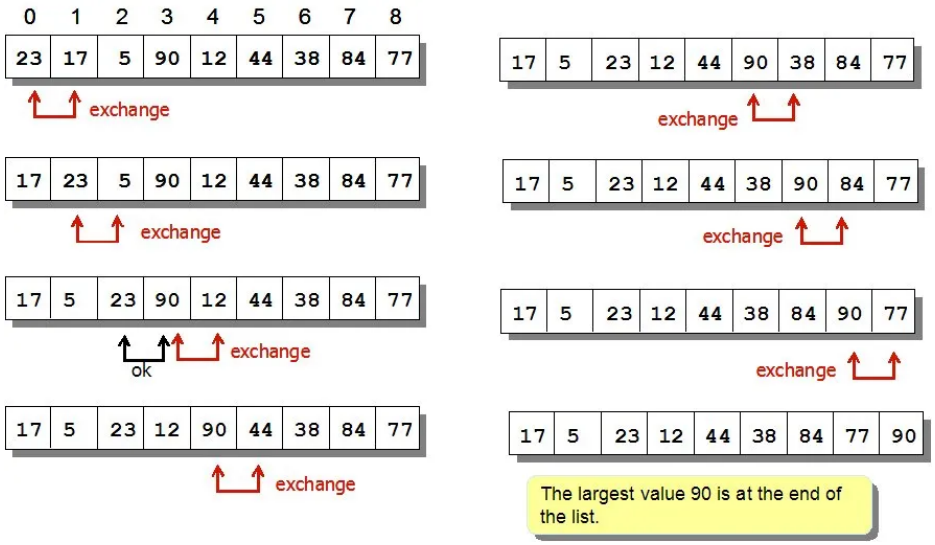
```
def separar_lista(lista):
    pares = []
    impares = []

    for i in lista:
        if i % 2 == 0:  # pares

            pares.append(i)
        else:
            impares.append(i)
    pares.sort()
    impares.sort()
    return pares, impares


#programa
lista = [4,5,6,7,1,5,2,8,12,15]
l1,l2= separar_lista(lista)
print(l1)
print(l2)
```

Prueba del programa:
[2, 4, 6, 8, 12]
[1, 5, 5, 7, 15]

Algoritmos de ordenamiento: El método más utilizado para ordenar listas el método bubble sort, un método que consiste en el ordenamiento de forma ascendente valores numéricos o strings el mismo procedimiento que resuelve en la función sort()

Consiste en intercambiar los valores menores que estén a la derecha del mayor. La siguiente figura ilustra el proceso a seguir.

Ilustración 16. Método bubble sort



Fuente: Ordenamiento Burbuja, https://yoandroide.xyz/ordenamiento-de-burbuja-bubble-sort-en-java/

Para entender cómo funciona este método revisemos el algoritmo:

```
def bubbleSort (p):
   tam = len(p)-1        #tam representa la longitud de la lista -1
   #print('tam =', tam)
   for i in range (0,tam):          #for externo
       for j in range (0, tam):     #for interno
           if (p[j] > p[j+1]):      # condición para intercambiar
               temp = p[j] # variable temp para almacenar valor actualizado
               p[j]=p[j+1]          # se produce el intercambio
               p[j+1]=temp          # se almacena temp en la posición
   return p
#Programa
x=[9,8,7,6,-5,5]
w= bubbleSort(x)
for a in w:
   print(a,end=" ")
```

Nótese que se utilizan ciclos anidados con el objetivo de comparar los valores enteramente. La función realiza el intercambio de valores cuando encuentra que un valor de la posición j es mayor que el siguiente valor el de j+1. Este proceso se hace por cada valor de la lista.

Referencias Bibliográficas

Luján J, (2019), Aprendre a Programar con Python, AlfaOmega, ISBN: 9789587786026

Hernández M., Bquero L., (2021), Estructura de Datos - Fundamentos Prácticos, Ediciones de la U, ISBN: 9789587922707

Betancourt J, Polanco I, (2021), 115 Ejercicios Resueltos de Programación C++, Ediciones de la U, ISBN: 9789587922974

Castaño A. - Python fácil EDICIÓN 2DA., ISBN - 978-2-7460- LOS FUNDAMENTOS DEL LENGUAJE DE PYTHON 3

Rodríguez L. - Libro digital. Version 2.5 2016, Python Programación, Escuela Superior Politécnica del Litoral

Python. Notes for Professionals. Python®. Notes for Professionals, GoalKicker.com – Python® Notes for Professionals 185

Programación desde cero, 2018, Ejercicios resueltos de funciones en Python, http://patriciaemiguel.com/ejercicios/python/2019/03/10/ejercicios-funciones-python.html#:~:text=El%20siguiente%20programa%20deber%C3%ADa%20imprimir,%C2%BFQu%C3%A9%20hay%20que%20corregir%3F&text=Soluci%C3%B3n%3A%20Las%20funciones%20no%20utilizan,las%20variables%20globales%20x%2C%20y.

Condor E.,2020, Algoritmos resueltos con Python, Editorial EIDEC, https://www.editorialeidec.com/wp-content/uploads/2020/10/Algoritmos-resueltos-con-Python.pdf, DOI: https://doi.org/10.34893/6kbn-5a63

El libro de Python, 2022, Uso del If en Python, https://ellibrodepython.com/if-python#uso-del-if

Meza J. 2021, Uso de expresiones anidadas, https://www.programarya.com/Cursos/Python/Condicionales/Condicionales-anidados-elif

Báez L. Algoritmos y estructura de datos, 2020, http://www.luchonet.com.ar/aed/?page_id=209