UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA

FACULTAD DE INGENIERÍA EN SISTEMAS DE INFORMACIÓN

MATEMÁTICA DISCRETA

ING. Melvin Cali

Proyecto Final

Luis Adrían Morales Alvarez
7690-25-5110
Santiago Alfonzo Benjamin Molina Balan
7690-25-4340
Johnny Brandon Escobar Garcia
7690-25-1502

SECCIÓN “A”

GUATEMALA OCTUBRE DE 2025

1. Ámbito de variables

1.1 Conceptos básicos de programación orientada a objetos (OOP o POO)

Los lenguajes de programación modernos como C#, Java, Utilizan, entre otros, el paradigma de programación orientada a objetos. En este paradigma, los programas se modelan en torno a objetos que aglutina toda la funcionalidad relacionada con ellos. La POO puede resultar confusa para mucha gente al principio, cuando se entra en contacto con ella.

Los lenguajes de programación antiguos, como C, Basic o COBOL, seguían un estilo procedimental a la hora de escribir código. Es decir, los programas escritos en estos lenguajes consisten en una serie de instrucciones, una detrás de otra, que se ejecutaban paso a paso. Para "encerrar" funcionalidades y poder reutilizar definían procedimientos (también llamados subrutinas o funciones), pero se usaban datos globales y era muy complicado aislar los datos específicos uno de otros. Podríamos decir que este tipo de lenguajes se centran más en la lógica que en los datos.

Sin embargo, los lenguajes modernos com C#, java o.. en realidad casi cualquiera, utiliza otros paradigmas para definir los programas. Entre éstos, el paradigma más popular es el que se refiere a la Programación Orientada a Objetos o POO.

El primer concepto y más importante de la POO es la distinción entre clase y objeto.

Una clase es una plantilla. Define de manera genérica cómo van a ser los objetos de determinado tipo. Por ejemplo, en un juego, una clase para representar a una persona puede llamarse Persona y tener una serie de atributos como Nombre, Apellido o Edad (que normalmente son propiedades), y una serie de comportamientos que pueden tener, como Hablar(), Caminar() o Comer() y que se implementan como métodos de la clase (funciones).

Una clase por sí sola no sirve de nada, pues no es más que un concepto, sin entidad real. Para poder utilizar una clase en un programa lo que hay que hacer es instanciar la. Instanciar una clase consiste en crear un nuevo objeto concreto de la misma. Es decir, un objeto es ya una entidad concreta que se crea a partir de la plantilla que es la clase. Este nuevo objeto tiene ya "Existencia" real, puesto que ocupa memoria y se puede utilizar en el programa. Así un objeto puede ser una persona que se llama Cristian López, de 37 años y que en nuestro programa podría hablar, caminar o comer, que son los comportamientos que están definidos en la clase.

1.2 Clases y objetos:

Clases: En la programación orientada a objetos, el modelado de la realidad se realiza a partir de clases. Las clases son modelos o abstracciones de la realidad que representan elementos de un conjunto de objetos como aviones, animales, personas, autos, computadoras, ecuaciones, figuras geométricas, elementos de una interfaz gráfica, archivos, etc.

Una clase de programación orientada a objetos es una plantilla o modelo utilizado para definir objetos, que son instancias de esa clase.

En las clases, la estructura y el comportamiento se definen especificando los métodos (funciones) y los atributos (variables). Las variables contienen los datos y las funciones ejecutan acciones específicas sobre ellos. Una vez creada una clase, se pueden crear varios objetos de la misma clase, pero cada uno tiene un conjunto diferente de atributos y comparte los mismos métodos.

Las clases proporcionan una estructura clara para los programas y ayudan a organizar y gestionar el código eficazmente. Al combinar datos y métodos relacionados, las clases promueven la modularidad en todo el programa.

Además, las clases admiten conceptos clave de OOP como herencia y encapsulación, lo que garantiza la reutilización del código y la restauración del programa.

Objetos: Los objetos en programación representan cosas del mundo real, así como conceptos abstractos con sus características y comportamientos específicos. Un objeto cuenta con su estructura interna que combina variables, funciones y estructuras de datos. Usando el nombre del objeto y la sintaxis según el lenguaje de programación, puedes visualizar los valores del objeto y llamar las funciones que tiene predefinidas.

Los elementos de un objeto se dividen en dos categorías principales: propiedades y métodos.

Las propiedades, también conocidas como atributos, incluyen información sobre el objeto. Por ejemplo, si consideramos un objeto Coche, algunas de sus propiedades serán: el color, la marca, el modelo o el año de fabricación.

Los métodos definen las operaciones que se pueden realizar con el objeto. Por ejemplo, para el objeto Coche, los métodos podrían ser acelerar, frenar o girar.

1.3 Métodos

Un método es una función definida dentro de una clase que describe el comportamiento de los objetos de esa clase. Los métodos permiten a los objetos realizar tareas, modificar atributos, interactuar con otros objetos o retornar valores.

Los métodos pueden ser:

- **Método de instancia:** Actuando sobre los atributos de un objeto en particular.
- **Método estático:** No necesita una instancia de la clase y puede ser llamado directamente desde la clase.

Estructura básica de un método

Los métodos generalmente siguen esta estructura:

1. **Nombre del método:** Identificador único para invocarlos.
2. **Parámetros:** Datos que el método necesita para realizar su tarea.
3. **Cuerpo del método:** El bloque de código que define la acción o tarea.
4. **Valor de retorno:** El resultado del método (opcional).

1.3 Atributos

Los atributos son la información que guarda en un momento determinado cada objeto que se instancia en una clase. Cada objeto tiene un espacio de memoria para cada atributo que se presenta en clase. Esta información, en cada momento, se conoce como estado del objeto.

Para determinar cuáles serían los atributos de una clase y de los objetos derivados de l a misma, tenemos al menos cuatro fuentes que pueden generarse:

- Entradas y Salidas.
- Variables intermedias: Que puedan considerarse importantes mantener para no calcularlas en cada momento dada su interdependencia en la solución del problema.
- Constantes: como el valor de pi o el número máximo de intentos que puede introducir una contraseña o el número máximo de inscripciones que puede tener un alumno o colores o nombres de los días de la semana.

La relación que permite incorporar atributos a una clase se conoce como “agregación” e indica que una clase u objeto se integra por atributos que pueden ser datos primitivos y también pueden ser objetos de una clase.

Bajo el segundo principio de la programación orientada a objetos todos los atributos no deben ser accesibles desde el exterior de la clase u objeto para recuperar su valor o modificarlo, es decir deben tener el modificador private. Por eso, debe agregar el

modificador private a Todos los atributos. Esta propiedad se conoce como encapsulamiento.

En programa de cómputo, al menos la información que se conoce como datos entradas y salidas o resultados deberían ser como atributos.

Se tendrá acceso a la información almacenada en las variables o atributos de las entradas y las salidas utilizando métodos getters (obtener) y setters(asignar) información a partir del segundo principio de la programación orientada a objetos conocido como encapsulamiento.

1.4 Encapsulamiento

El encapsulamiento o encapsulación en programación es un concepto relacionado con la programación orientada a objetos. y hace referencia al ocultamiento de los estados internos de una clase al exterior. Dicho de otra manera, encapsular consiste en hacer que los atributos o métodos internos a una clase no se puedan acceder ni modificar desde fuera, si no que tan solo el propio objeto pueda acceder a ellos.

Para la gente que conozca Java, le resultará un término muy familiar, pero en Python es algo distinto. Digamos que Python por defecto no oculta los atributos y métodos de una clase exterior.

La encapsulación es un mecanismo para reunir datos y métodos dentro de una estructura ocultando la implementación del objeto, es decir, impidiendo el acceso a los datos por cualquier medio que no sean los servicios propuestos. La encapsulación permite, por tanto. garantizar la integridad de los datos contenidos en el objeto. Por lo tanto, si queremos proteger la información contra modificaciones inesperadas, debemos recurrir al principio de encapsulamiento.

La encapsulación permite definir niveles de visibilidad para los elementos de la clase. Estos niveles de visibilidad definen los derechos de acceso a los datos en función de si se accede a ellos mediante un método de la propia clase, heredada o de cualquier otra clase. Hay cuatro niveles de visibilidad:

- **Visibilidad por defecto:** No se especifica ningún modificador de visibilidad.
- **Visibilidad pública:** Las funciones de todas las clases pueden acceder a los datos o métodos de una clase definida con el nivel de visibilidad <<público>>. Este es el nivel más bajo de protección de datos.
- **Visibilidad protegida:** El acceso a los datos está restringido a las funciones de las clases heredadas, es decir, por las funciones miembro de la clase y las clases derivadas. Así, un atributo o un método declarado <<protegido>> es accesible sólo a las clases de un paquete y sus subclases, aunque estén definidas en un paquete diferente.
- **Visibilidad Privada:** El acceso a los datos se limita a los métodos de la propia clase. Este es el nivel más alto de protección de datos.

1.5 Herencia

La herencia permite que se puedan definir nuevas clases basadas de unas ya existentes a fin de reutilizar el código, generando así una jerarquía de clases dentro de una aplicación. Si una clase deriva de otra, esta hereda sus atributos y métodos y puede añadir nuevos atributos, métodos o redefinir los heredados.

En Java tenemos que tener claro cómo llamar a la clase principal de la que heredamos y aquella que hereda de ella, así, clase que se hereda de su superclase. La clase heredada se llama subclase. Por lo tanto, una subclase es una versión especializada de

una superclase. Hereda todas las variables y métodos definidos por la superclase y agrega sus propios elementos únicos.

Terminología importante:

- **Superclase:** La clase cuya característica se heredan se conoce como superclase (o una clase base o una clase principal).
- **Subclase:** La clase que hereda la otra clase se conoce como subclase (o una clase derivada, clase extendida o clase hija). La subclase puede agregar sus propios campos y métodos, además de los campos y métodos de la superclase.
- **Reutilización:** La herencia respalda el concepto de "reutilización", es decir, cuando queremos crear una clase nueva y ya hay una clase que incluye parte del código que queremos, podemos derivar nuestra nueva clase de la clase existente. Al hacer esto, estamos reutilizando los campos/atributos y métodos de la clase existente.

2. Estructuras de datos básicas

Las estructuras de datos son estructuras o métodos que se usan para almacenar datos dentro de las mismas.

La creación de estructuras representa la organización de almacenamiento de los datos en la memoria del dispositivo, creando reglas del cómo se accede, modifica y se procesa los datos durante la ejecución de programas. Una estructura de datos es una forma particular de organizar datos en un ordenador para que puedan ser utilizados de manera eficiente. La elección correcta de la estructura determina no solo la legibilidad del código, sino también su rendimiento y escalabilidad. Cada estructura ofrece diferentes ventajas y limitaciones. Algunas permiten acceso rápido a elementos específicos, mientras otras facilitan la inserción y eliminación de datos. Comprender estas características nos permite tomar decisiones informadas según las necesidades específicas de cada problema. (Sastre, 2025)

Clasificación

Las estructuras de datos se clasifican por lo general en dos categorías:

Estructuras primitivas: Los tipos de datos primitivos representan los elementos más básicos que puede manejar un lenguaje de programación. Incluyen enteros, flotantes, caracteres, lógicos y entre otros. Son tipos de datos soportados por el hardware y la forma tradicional de los mismos datos.

Estructuras no primitivas: Son datos que son creados a partir de datos primitivos o no primitivos. Siendo estos datos que se dividen en dos tipos. Lineales y no lineales.

Estructuras lineales: Son estructuras que se organizan de manera secuencialmente lineal, donde cada dato o elemento en una estructura de datos tiene un predecesor y un sucesor, excepto el primero y el último elemento.

Los arreglos y listas representan la estructura lineal más fundamental. Almacenan elementos del mismo tipo que el de todos los elementos en posiciones de la memoria, permitiendo acceso directo mediante filas, columnas o, individualmente, índices.

Las listas extienden el concepto de los arreglos permitiendo modificar su tamaño durante la ejecución.

Pilas y colas. Las pilas o stacks implementan el principio LIFO (Last in, First Out). Los elementos que se introduzcan dentro de la pila o que se eliminen únicamente desde un extremo llamado límite o "tope". Las colas o queues, siguen el principio de FIFO (First In, First Out). Los elementos que se introduzcan en esta estructura de datos entran por un extremo y se eliminan los elementos del otro extremo, simulando de esa manera una cola o fila.

Estructuras no lineales: Las estructuras de datos no lineales se ordenan no según el orden de ingreso o un índice, sino, que de forma jerárquica o en red, donde los elementos almacenados pueden ser reemplazados o reemplazar a múltiples elementos dentro de la misma.

Los árboles representan una jerarquía en forma de árbol donde cada elemento o nodo puede tener elementos que dependen del mismo llamados hijos, a su vez también los nodos no pueden tener hijos. El nodo superior o principal es el nodo que se encuentra en la cima de todos los nodos y tiene como nombre raíz, en cambio, los nodos que no tienen hijos como tal, son denominados hojas.

Los grafos son una estructura de datos que son más flexibles que los árboles, teniendo una estructura más general y conectando los nodos, esta vez denominados vértices, conectados por medio de aristas. Estos mismos pueden representar rutas de carretera, de conexiones de red, cableado, etc.

Operaciones para las estructuras de datos

A pesar de que las estructuras de datos tengan diferencias fundamentales respecto a su organización, funciones, jerarquías y otras características esenciales, las operaciones que se realizan para cada estructura son las mismas para todas.

Añadir: Se añaden nuevos elementos dentro de la estructura.

Eliminación: Se eliminan elementos.

Búsqueda: Localizar los elementos requeridos.

Recorrido: Se leen o se visualizan todos los elementos que se encuentran dentro de la estructura de datos.

Actualización: Se modifican o editan los elementos dentro de la estructura de datos.

A pesar de que las mismas operaciones son válidas y pueden realizarse dentro de una estructura, las funciones y operaciones, como en su forma de aplicarse y cómo funcionan pueden variar según la estructura de datos.

3. Listas

Las listas, en programación, son una lista de datos que se almacenan de forma ordenada y secuencial. Las listas tienen la capacidad de almacenar los datos y asignarles un alias para identificarse y acceder a ellos por medio de un índice. Otra característica de las listas es su capacidad de almacenar distintos tipos de datos, de ahí su flexibilidad. (apinem web, 2025)

Las ventajas de usar listas como estructura de datos son las siguientes:

1. **Organización:** Las listas ayudan a organizar la información dentro de ella. Con la característica de poder identificar y localizar los datos por medio de índices.
2. **Acceso rápido:** Se puede localizar directamente los datos requeridos dentro de la lista sin necesariamente leer todos los datos secuencialmente.
3. **Almacenamiento de información:** Las listas pueden almacenar varios tipos de datos, por lo que puede incluso almacenar información legible y entendible.
4. **Automatización:** Las tareas repetitivas pueden realizarse con listas.

Características de las Listas

1. **Ordenados:** Las listas son ordenadas y sus datos siguen un orden.
2. **Indexado:** Cada elemento está identificado por medio de un índice por el cual se puede acceder directamente.
3. **Mutables:** Las listas son mutables. Es decir, se puede modificar los elementos dentro de las mismas.

4. **Homogéneas o heterogéneas:** Las listas pueden almacenar desde solo un tipo de dato (homogéneas) o diversos tipos de datos (heterogéneas).

5. **Capacidad dinámica:** La capacidad de las listas son dinámicas. Es decir, pueden aumentar o disminuir su tamaño.

6. **Operaciones directas:** Gracias a su sistema de indexado, las operaciones que se realizan para un índice o un elemento se pueden realizar de forma directa sin realizar operaciones extras para llegar al dato a lo largo de una lista.

7. **Versatilidad:** Las listas pueden representar varios casos para diversas tareas, por lo que son útiles en diversas tareas.

8. **Facilidad de iteración:** Las listas son fáciles de iterar. Es decir, son fáciles de recorrer todos sus elementos por medio de bucles.

9. **Posibilidad de listas anidadas:** Las listas pueden contener otras listas dentro de las mismas. Por lo que puede dar paso a listas complejas para mayores tareas.

Operaciones de acceso a una lista

El acceso a elementos dentro de una lista permite recuperar elementos en específicos. Las siguientes operaciones son:

1. **Indexado desde 0:** El primer índice de una lista siempre inicia desde 0.

2. **Acceso a elementos individuales:** Para acceder a un elemento de la lista, se utiliza como referencia el índice asignado a ese elemento.

3. **Indexado negativo:** Algunas listas en lenguajes de programación permiten un indexado negativo. Por ejemplo: al ingresar un índice -1, se toma el último elemento de la lista, -2 para el penúltimo elemento, y así sucesivamente hasta que excede.

4. **Operaciones de rebanado o Slicing:** El acceso no solo puede ser individual, se pueden crear variables para un rango de listas, o subconjuntos. Se crea una lista aparte para almacenar los datos extraídos de una ya existente.

5. **Manejo de índices fuera de los límites:** Acceder a índices que no tienen valores puede resultar en errores, como en la falla del programa o de extraer valores “basura” de la memoria.

4. Diccionarios

Un diccionario es una colección no ordenada que permite asociar índices con elementos. Con el índice del valor, se puede recuperar o extraer los valores con un índice en cuestión. El funcionamiento de un diccionario es similar al de las listas. En las listas se usa un índice que se asigna según el orden en el que se han asignado. En cambio, en diccionarios, el índice puede ser un entero, flotante, caracter, cadena, lista, entre otros. De igual manera, los índices tienen que ser únicos. (Programación UTFSM, 2025)

4.1. Tuplas

Una tupla es un grupo de elementos que, en programación, contiene elementos y valores ya definidos dentro del código fuente y pueden ser extraídos o leídos durante la ejecución. Sin embargo, las tuplas, a diferencia de las listas, diccionarios y otros tipos de agrupación de elementos dentro de una variable, los elementos de las tuplas son inmutables. Es decir, una vez definidas en el código fuente, los elementos dentro no pueden ser modificados, eliminados o añadir valores. Las tuplas sirven como una secuencia de valores heterogéneas que se usan para evitar errores que puedan alterar los valores de un conjunto de elementos que no se esperan ser alterados.

4.1.1. Características de una tupla:

Las tuplas tienen muchas similitudes con otros tipos de listados o conjuntos de valores, pero difiere en algunas funciones fundamentales:

- **Orden:** Los elementos de una tupla están almacenados en un orden específico. A diferencia de otros conjuntos de elementos, las tuplas tienen un orden establecido según el orden al que se ha añadido el elemento.

- **Heterogénea:** Los valores que se añaden dentro de las tuplas pueden ser de distintos tipos de datos como enteros, flotantes, lógicos y demás.

- **Acceso directo:** Se puede acceder de forma directa sin tener que leer o buscar secuencialmente cada elemento dentro de una tupla, por medio de un índice que se asigna según el orden en que se hayan introducido.

- **Inmutable:** Las tuplas, previamente mencionadas, una vez asignado sus valores, durante la ejecución del programa no se pueden alterar o modificar.

- **Rebanado:** Al igual que en otros conjuntos de elementos, los elementos de las tuplas pueden extraerse en rangos y almacenarlos en otra tupla o conjunto de elementos. Llamados Sub-tuplas.

- **Conversión:** Las tuplas pueden convertirse en otro tipo de conjunto de datos y viceversa. (Isaac, 2025)

4.2. Conjuntos

Los conjuntos son grupos de datos que, pueden almacenar varios tipos de estas según su tipo de datos o el propósito de estas. En programación, los conjuntos son datos agrupados que, estos datos, están desordenados y sin repetición, similares a los conjuntos matemáticos. Las operaciones que se pueden hacer para los mismos son operaciones sobre los elementos y operaciones sobre los conjuntos. (Cátedra de Algoritmos y Programación II, 2025)

4.2.1. Características de un conjunto:

Las características de los conjuntos son similares a otras estructuras de datos, pero con diferencias fundamentales a sus funciones:

- **Datos únicos:** Los datos almacenados dentro de un conjunto son únicos y no se puede repetir, evitando de esa forma un posible almacenaje de datos repetidos de forma innecesaria.

- **Operaciones definidas:** Las operaciones para los conjuntos ya están definidos en comandos reservados, por lo que se pueden hacer operaciones entre conjuntos y sus elementos como unión, intersección, negatividad, producto, etc.

- **Indexado por Hash:** Los elementos de un conjunto están indexados por el método de la tabla Hash o índice. Por lo que se puede acceder a un valor en específico sin tener que recorrer todos los valores hasta el requerido. (Fundamentos, s.f.)

4.3. Diferencias entre una tupla y un conjunto

Tabla 1

Diferencias entre conjuntos y tuplas.

Conjuntos	Tuplas
En los conjuntos, los elementos no se pueden repetir, por lo que son únicos.	En las tuplas, los elementos se pueden repetir, serán sus índices para poder identificarlos
Los conjuntos son mutables, por lo que pueden alterarse sus valores.	Las tuplas son inmutables, por lo que no se pueden modificar durante la ejecución.
El orden en los conjuntos es inexistente, por lo que están desordenados.	El orden en las tuplas se rige según el orden de los valores agregados y su índice.
Con los conjuntos se puede realizar varias operaciones con comandos reservados.	Con las tuplas, las operaciones tienen que hacerse sin varios comandos reservados.

Nota: Los conjuntos tienen como función principal que los datos no se repitan y poder tener una lista de datos con el que se pueden operar sin tener el riesgo de repetir innecesariamente

los mismos datos. En cuanto a las tuplas, su principal función es evitar que los datos durante la ejecución del programa no se alteren por algún error de programación.

4.4. Funciones y modularidad

Las funciones y modularidad en programación se refieren a dividir el código en distintas funciones o bloques de comando que se ejecutarán cuando el programa lo pida. En vez de realizar todo el código y sus acciones en una sola secuencia de instrucciones, se realizan funciones, bloques de comando, o módulos, que se ejecutarán según cuando se requiera. Una forma de ahorrar, esclarecer y estructurar el código de una manera más eficiente y entendible.

Las funciones se definen por un nombre de variable y luego por parámetros, los cuales serán el input y los valores que entran dentro de la función y pasan por su debido proceso. En cuanto a modularidad se refiere, se trata de los mismos bloques de las funciones o, de otro tipo de funciones que se guardan en otras fuentes de código, siendo las librerías las cuales sirven como plataforma para acceder a datos, procesos o incluso otras funciones anidadas.

4.4.1. Funciones

Las funciones son bloques de código que realizan las instrucciones que se encuentran dentro de ella de manera secuencial. Sin embargo, las funciones se ejecutan cuando se les llama dentro del código fuente y no cuando el código pase por la misma al ejecutarse. En adición, las funciones existen los parámetros. Los parámetros son datos que sirven como input dentro de una función. Es decir, que los valores que se asignan a los parámetros son aquellos que se obtienen ya sea de forma directa (asignación directa) u obteniendo los valores durante la ejecución del programa.

Dentro de las funciones también se puede llamar a otras funciones y asignar valores a sus parámetros.

4.4.1.1. Características de las funciones:

Algunas de las características de las funciones son:

- **Automatización:** Las funciones pueden contener procesos e instrucciones que pueden realizarse cada vez que el programa lo llame por su nombre de variables, por lo que, dependiendo de las tareas, se puede automatizar operaciones repetitivas.

- **Reutilización:** Las funciones pueden ser reutilizadas las veces que se deseen dentro de un programa. Como, por ejemplo, el proceso de una calculadora para realizar operaciones. Realizar los procedimientos de cálculo al recurrir a la o las funciones necesarias y utilizar los inputs en los parámetros.

- **Legibilidad mejorada:** Al usar funciones estas pueden ser organizadas a lo largo del programa sin tener que repetir los mismos procesos estrictamente lineales y repetir innecesariamente los códigos.

- **Flexibilidad:** La flexibilidad que maneja las funciones permite que las instrucciones al ejecutarse puedan realizarse cuando el programa lo requiera o cuando se desea.

- **Dinamismo:** Las funciones aceptan todo tipo de datos y pueden tener todo el tamaño que se necesite. Al fin al cabo, las funciones son en sí procesos

que se ejecutan al ser llamados sin tener que realizar procesos completamente lineales.

- **Estrictas:** Las funciones mantienen los procesos hasta que llegue al final de la ejecución de todas las instrucciones que contiene. Es decir, el código ejecuta las instrucciones de la función llamada por completo.

- **Seguras:** El programa, cuando se ejecuta una función, ejecuta todas las instrucciones dentro de ella y no es hasta que termina con todas las instrucciones que continúa con la secuencia del resto de código.

Las funciones, además, usan la variable del cual es el nombre de la función como el lugar de la memoria donde se almacenará el resultado que se decidirá devolver. Es decir, el resultado o la información que se desea devolver será el valor de la variable, la cual podrá ser almacenada en otra variable para que este sea usado por el programa.

4.4.2. Módulos

Los módulos son parecidos a las funciones. Sin embargo, los módulos, a pesar de comportarse similar a una función, sus usos son aún más amplios que las mismas funciones respecto a su acceso, su ubicación y cómo se aplican dentro de distintos programas o archivos de programas.

Las funciones son “pequeños” módulos que se encuentran dentro del mismo programa y únicamente ese archivo puede acceder a esa función. En cambio, los módulos, estos no se encuentran como tal en el mismo archivo que el programa y su ejecución, estos se encuentran en un archivo aparte y se construye dentro de este “programa o script modular”, las funciones, datos o información para que el archivo que ejecute el programa pueda acceder a estos mismos, una clase de lista de datos, pero más amplia y con mayor flexibilidad que las

diversas estructuras de datos que hay. Hacer uso de ellos. Para ello, existe dos tipos de módulos.

Módulos de datos: Los módulos de datos son aquellos archivos modulares que se usan exclusiva y únicamente para obtener datos de ella. Un archivo puede llamar al archivo modular para solicitar los datos que se piden en el archivo del programa.

Módulos de funciones: Los módulos también pueden contener funciones. Cuando se crea un archivo modular para almacenar una función o varias para operaciones matemáticas, el archivo del programa puede solicitar los datos de la variable de la función y ejecutar la función.

4.4.2.1. Características de un módulo

Los módulos pueden tener ciertas características que la difiera de una función como tal:

- **Reutilización:** Los módulos pueden ser reutilizados las veces que requiera el programa. Sin embargo, el acceso a estos módulos es diferente al de una función.
- **Acceso:** Los módulos se crean a partir de "librerías" o archivos modulares donde únicamente los módulos rigen el contenido de estos. Por lo que todo programa que logre ubicar y pueda hacer uso del contenido de los módulos podrá extraer información llamando no a una función, sino a un request y pedir los datos según el mismo nombre de las variables que se encuentran en el módulo y almacenarlos en variables que se crean en el archivo del programa.

- **Dinamismo:** Al igual que las funciones, los módulos no tienen como tal un límite para datos o información, por lo que se puede almacenar cualquier tipo de dato, función o proceso dentro de una.

- **Disponibilidad:** Los módulos al estar en un archivo aparte, los programas que puedan acceder a ella pueden extraer su información. A diferencia de una función que, únicamente el programa en donde está puede hacer uso de ella.

- **Integro:** Los módulos conservan los datos e información que se asigna a variables, por lo que sin importar si el archivo del programa cambie su estructura, al llamar correctamente las variables del archivo módulo, conserva y transmite los datos exactos tal como han sido ingresados u obtenidos.

4.5. Aprender a cómo crear una función

Para crear una función para que el código pueda tener una mejor organización y por ende una mejor función, se debe considerar primero 3 factores:

1. **Tareas repetitivas:** Las funciones son sumamente útiles para tareas que se deban hacer muchas veces.

2. **Complejidad del programa:** Si se trata de un programa que deba tener muchas operaciones manejadas por el usuario, las funciones pueden organizar mejor el código sin tener que reescribir el código en las partes donde se deba realizar las operaciones en cuestión.

3. **Claridad del programa:** El factor de realizar un programa y que esté siendo codificado por varias personas, las funciones ayudan a guiar y señalar las instrucciones para las personas que operan con el mismo programa.

Por ejemplo: se planea crear un programa que pueda calcular los catetos e hipotenusa de un triángulo rectángulo. Una solución puede ser el crear muchas condicionales, una sobre otra, para poder especificar lo que pide el programa y el usuario, el ingreso de datos y el cálculo para conseguir lo deseado. Sin embargo, esto puede saturar al programa de líneas de código innecesariamente repetidas a lo largo del programa, confundir a las personas al no tener un programa claro de entender y que su optimización sea mala.

Una solución para mejorar el programa sería usar funciones. Las operaciones pueden ser almacenadas en funciones por separado según lo requiera el usuario y estas ser llamadas por el programa para ser ejecutadas y usando los valores introducidos, como las medidas del triangulo conocidas, en los parámetros. Y únicamente usando condicionales para crear las opciones sobre lo que el usuario desea calcular.

5. Funciones:

5.1. Funciones simples:

Las funciones simples son funciones que contienen instrucciones sencillas, lineales y que su única función fundamental es ejecutar, realizar las instrucciones y devolver en la variable un valor.

Por ejemplo: Una función simple sería colocar en una misma una instrucción que imprima en la terminal “Hola Mundo”. Esto por sí no se ejecutará como tal hasta que el programa lo llame por su nombre de variable seguido de dos paréntesis, lugar donde van los parámetros que sirven como input, pero para la función del ejemplo no lo requiere, pero si los paréntesis. Esto solo hará que se ejecute como tal las instrucciones de la función. Otro ejemplo sería tener líneas de instrucción dentro de la función para sumar dos números. En adición, el programa contendrá otras líneas de código fuera de la función que le pedirá al usuario ingresar dos números enteros o flotantes, leerá los números que sean ingresados, luego llamará a la función y esta vez, como la función está definida por dos parámetros o variables, entonces se debe colocar la misma cantidad de variables para que se ejecute como mínimo, entonces, se colocará las variables donde se almacenó los valores ingresados. De esta manera, la función tendrá como input los valores ingresados y realizará las instrucciones con esos valores. De último, el total sumado se almacenará en una variable. Este no debe ser un parámetro. Y el valor que regresará (o almacenará en la variable que tiene la función como nombre) el valor de esa variable que se decida devolver.

Hay que recordar que todas las funciones deben devolver un valor a su variable, a pesar de que sea una función que no se espera que devuelva un valor necesario. Por lo que se coloca

un cero o cualquier número entero para que esta condición de valor devuelto sea verdadera y ejecute o finalice el programa sin ningún problema.

5.2. Parámetros

Los parámetros son el input de las variables que se usarán en las instrucciones dentro de una función. Esto es la parte fundamental de la flexibilidad y utilidad de las funciones ya que, estos parámetros pueden ser cualquier número o valor en cuanto el programa llame a la función.

Los parámetros funcionan cuando el programa llama a una función y con sus respectivos parámetros o variables. Estas variables deben tener o ser el mismo tipo de dato para que funcione y sea válido para las operaciones. Los valores con los cuales fue llamado la función serán los utilizados para realizar las instrucciones dentro de la función, por lo que hay que tomar en cuenta que, si una variable es declarada o en si está declarada inicialmente en una función, esa variable únicamente será válida para las instrucciones de esa función.

5.3. Valor de retorno

El valor de retorno es el valor final que la variable (el nombre de la función) tendrá cuando se finalice o se llegue a la línea de *return* seguido del valor o la variable con el valor.

Por ejemplo, se tiene una función que realiza la operación de sumar dos números y dentro de las instrucciones de esta se cuenta con una variable que sirve para almacenar el valor resultado de la suma, es entonces que esa variable se coloca después de la palabra reservada *return*. El valor de la variable del resultado será la que retornará y se almacenará

temporalmente en el nombre de la función, para luego ser almacenada fuera de la función en otra variable que contendrá el valor retornado.

6. Ámbito de variables

Ámbito local: Es aquella cuyo ámbito se restringe a la función que la ha declarado se dice entonces que la variable es local a esa función. Esto implica que esa variable sólo va a poder ser manipulada en dicha sección, y no se podrá hacer referencia fuera de dicha sección. Cualquier variable que se defina dentro de las llaves del cuerpo de una función se interpreta como una variable local a esa función.

Las variables locales son accesibles desde su declaración hasta el final del bloque de código del fichero en el que han sido declaradas.

Un bloque de código viene determinado por una pareja de llaves ({}).

Si el bloque donde se ha declarado la variable local contiene a su vez otro bloque, también es accesible dentro de ellos.

En caso de que una variabilidad parezca declarada en varios niveles de anidamiento dentro de un bloque, prevalece la declaración del bloque más interno.

Las variables locales se destruyen, dejan de ser accesibles, cuando la ejecución del programa abandona el bloque donde han sido declaradas.

Cuando una variable x es local a una función func1, significa que la función func1 es la propietaria de dicha variable, y puede acceder a ella y modificarla. Si cualquier otra función del programa necesita conocer el valor de la variable x, es la función func1 la que debe transferir el valor de x a través del paso de argumentos en la llamada a la función. Si además esta función desea modificar el valor de dicha variable, entonces tendrá que devolver el nuevo valor a func1, y será func1 quien se encargue de asignar el valor devuelto a su variable x.

Ámbito global: Es aquella que se define fuera del cuerpo de cualquier función, normalmente al principio del programa, después de la definición de los archivos de biblioteca (#include),

de la definición de constantes simbólicas y antes de cualquier función. El ámbito de una variable global son todas las funciones que componen el programa, cualquier función puede acceder a dichas variables para leer y escribir en ellas. Es decir, se puede hacer referencia a su dirección de memoria en cualquier parte del programa.

Las variables globales son accesibles por todas las funciones desde su declaración hasta el final del fichero.

El uso de variables globales no es aconsejable a pesar de que parezca útil y cómodo:

- Disminuye la legibilidad
- Su uso puede producir efectos colaterales, al producirse alteraciones no previstas de su valor en una parte del programa que afecta a su uso en otra zona.
- Dificulta la modularidad del código.

Las variables y/o funciones globales se justifican cuando se necesitan en casi todas las funciones del programa.

Las variables std::cin y std::cout, son globales.

Las variables declaradas dentro de una función son automáticas por defecto, es decir, sólo existen mientras se ejecuta la función. Cuando se invoca la función se cran estas variables en la pila y se destruyen cuando la función termina. La única excepción la constituyen las variables locales declaradas como estáticas (static). En este caso, la variable mantiene su valor entre cada dos llamadas a la función aún cuando su visibilidad sigue siendo local a la función.

7. Bibliografía

CampusMVP. (s.f.). Los conceptos fundamentales sobre programación orientada a objetos explicados de manera simple. https://www.campusmvp.es/recursos/post/los-conceptos-fundamentales-sobre-programacion-orientada-objetos-explicados-de-manera-simple.aspx

Cincom. (s.f.). Class in object-oriented programming. https://www.cincom.com/blog/smalltalk/class-in-object-oriented-programming/

UNAM - Portal Académico. (s.f.). Clases y objetos. https://portalacademico.cch.unam.mx/cibernetica1/analisis-y-diseno-en-poo/clases-y-objetos

EBAC. (s.f.). Qué es un objeto en programación. https://ebac.mx/blog/objeto-en-programacion

OpenWebinars. (s.f.). Introducción a POO en Java: objetos y clases. https://openwebinars.net/blog/introduccion-a-poo-en-java-objetos-y-clases/

Web Design Cusco. (s.f.). Métodos y constructores en la programación orientada a objetos. https://webdesigncusco.com/metodos-y-constructores-en-la-programacion-orientada-a-objetos-poo/

UNAM - Portal Académico. (s.f.). Atributos en la programación orientada a objetos. https://portalacademico.cch.unam.mx/cibernetica2/principios-programacion-orientada-a-objetos/atributos

El Libro de Python. (s.f.). Encapsulamiento en POO. https://ellibrodepython.com/encapsulamiento-poo

DataScientest. (s.f.). Encapsulación: definición e importancia. https://datascientest.com/es/encapsulacion-definicion-e-importancia

NTT Data. (s.f.). Herencia en programación orientada a objetos. https://ifgeekthen.nttdata.com/s/post/herencia-en-programacion-orientada-objetos-MCPV3PCZDNBFHSROCCU3JMI7UIJQ?language=es

Universidad de Valladolid. (s.f.). Ámbito de variables. https://www2.eii.uva.es/fund_inf/cpp/temas/2_tipos_variables/ambito_variables.html

Universidad de Granada. (s.f.). Capítulo 6: Tipos y variables. https://ccia.ugr.es/~jfv/ed1/c/cdrom/cap6/cap62.htm

Markdown.es. (s.f.). Sintaxis Markdown. https://markdown.es/sintaxis-markdown/