***Proyecto Final Compiladores***

*H.K. Alvarez Ibarra*

*9989-20-13495 Universidad Mariano Gálvez*

*Compiladores*

[*halvarezi@miumg.edu.gt*](mailto:halvarezi@miumg.edu.gt)

**Introducción**

En el ámbito de la informática y el desarrollo de software, la creación de un compilador y un lenguaje de programación representa un desafío significativo que combina teoría computacional, diseño de software y creatividad en ingeniería. Este proyecto se enfoca en el desarrollo de "Hasklyns", un lenguaje de programación único que incorpora elementos de varios videojuegos populares como Mario Bros, Dragon Ball Z, God of War, Naruto, The Last of Us, y Red Dead Redemption II. Hasklyns ha sido diseñado no solo para ser funcional y robusto, sino también para hacer que el proceso de programación sea más interesante y accesible para los entusiastas de los videojuegos.

El lenguaje Hasklyns tiene como objetivo principal simplificar la programación mientras se introduce a los programadores en un entorno sintáctico que les resulte familiar y emocionante. Al utilizar nombres y conceptos de personajes de videojuegos reconocidos para sus estructuras sintácticas, Hasklyns busca capturar la atención de los usuarios y ofrecer una experiencia de aprendizaje única. Desde declaraciones de variables hasta estructuras de control complejas, cada aspecto del lenguaje está pensado para reflejar la acción y aventura que caracterizan a los juegos que lo inspiran.

Para complementar el diseño de este lenguaje, se ha desarrollado un compilador específico para Hasklyns. Este compilador no solo interpreta y ejecuta instrucciones escritas en Hasklyns, sino que también optimiza el código y garantiza que se ejecuten de manera eficiente. La implementación de este compilador incluye características avanzadas como análisis sintáctico y semántico, generación de código intermedio y final, y manejo de errores para proporcionar feedback claro y útil a los desarrolladores.

**¿Qué es Hasklyns?**

Hasklyns es un lenguaje de programación innovador, diseñado con el propósito de hacer la codificación más accesible y atractiva para un público diverso, especialmente para los aficionados a los videojuegos. Este lenguaje se destaca por incorporar elementos y terminología de varios videojuegos populares, como Mario Bros, Dragon Ball Z, God of War, Naruto, The Last of Us y Red Dead Redemption II, utilizando los nombres y mecánicas de estos juegos para estructurar su sintaxis.

**Alfabeto de Hasklyns**

1. **Caracteres Alfabéticos:**
   * Letras mayúsculas (A-Z)
   * Letras minúsculas (a-z)
   * Esto permite palabras clave basadas en nombres de personajes y acciones, como **mario**, **kratos**, **survive**, **recover**, etc.
2. **Dígitos Numéricos:**
   * Números (0-9)
   * Útiles para declarar valores numéricos, definir iteraciones en bucles, especificar tamaños, etc.
3. **Símbolos Especiales:**
   * Signos de puntuación y operadores aritméticos y lógicos, como **=**, **+**, **-**, **\***, **/**, **%**.
   * Paréntesis **(** y **)**, utilizados para definir el orden de las operaciones y agrupar argumentos en llamadas de función.
   * Llaves **{** y **}**, que se utilizan para delimitar bloques de código.
   * Corchetes **[** y **]**, para definiciones de arrays o listas si se necesitan.
   * Punto y coma **;**, utilizado para terminar declaraciones.
   * Coma **,**, para separar elementos en listas o argumentos en funciones.
   * Puntos **.** para acceso a métodos o propiedades de objetos.
   * Signos de igualdad y comparación: **==**, **!=**, **<**, **>**, **<=**, **>=**.
   * Operadores lógicos: **&&** (AND), **||** (OR), **!** (NOT) en su versión simbólica.
4. **Caracteres de Espacio:**
   * Espacio en blanco, tabulación, y nuevas líneas, que ayudan a separar los elementos del código para mejorar la legibilidad y estructura.

Este alfabeto garantiza que los desarrolladores pueden expresar una variedad de operaciones lógicas y matemáticas, definir estructuras de datos, y escribir código claro y organizado. Al utilizar estos caracteres, Hasklyns se mantendrá accesible y familiar para los programadores, al tiempo que ofrece su única vuelta de tuerca temática inspirada en videojuegos.

**Declaración de Variables en Hasklyns**

Definición:

La declaración de variables en Hasklyns es el proceso mediante el cual se asigna un identificador y, opcionalmente, un valor inicial a una variable que luego se utilizará para almacenar datos dentro del programa. Hasklyns utiliza palabras clave inspiradas en personajes de videojuegos famosos para definir el tipo de datos de las variables, haciendo que el lenguaje sea más intuitivo y divertido para los aficionados a los videojuegos.

**Sintaxis:**

En Hasklyns, la declaración de variables sigue el formato:

<tipo> <nombre\_variable> = <valor\_inicial>;

El **<tipo>** es una palabra clave específica del videojuego que indica el tipo de datos de la variable.

Cómo se usa:

Las variables en Hasklyns se utilizan para almacenar datos que pueden cambiar durante la ejecución del programa. La elección de palabras clave para los tipos de datos está diseñada para resonar con los usuarios que son fans de videojuegos, asociando tipos comunes de datos con elementos específicos de juegos.

Declaración de variables usando **Python**

health = 100

player\_name = "Link"

ammo\_count = 30

**Ejemplos en Hasklyns**

// Declaración de variables usando Hasklyns

monedas health = 100; // Inspirado por Mario Bros, usando 'monedas' para representar la salud como algo que se puede coleccionar o perder.

heroe player\_name = "Kratos"; // Inspirado por God of War, utilizando 'heroe' para representar nombres de personajes.

balas ammo\_count = 30; // Inspirado por Grand Theft Auto IV, utilizando 'balas' para representar el conteo de municiones.

**Palabras Claves en Hasklyns**

Definición:

Las palabras clave en Hasklyns son identificadores reservados que tienen significados especiales definidos por el lenguaje. Estas palabras se utilizan para realizar operaciones específicas, controlar el flujo del programa, y definir estructuras de datos, entre otros.

Sintaxis:

Las palabras clave en Hasklyns se utilizan como parte de la sintaxis para estructurar el código de manera clara y lógica, siguiendo un formato inspirado en elementos y personajes de videojuegos.

Cómo se usa:

Cada palabra clave controla una parte específica de la lógica del programa. Por ejemplo, para iniciar un bucle, para evaluar condiciones, o para definir bloques de código.

Ejemplos en Python y Hasklyns:

**Python:**

if condition:

# do something

else:

# do something else

**Hasklyns:**

**Mario Bros para bucles (for):**

mario (inicio; condición; incremento) {

# acciones a repetir

}

**Dragon Ball Z para condiciones (if):**

goku (condición) {

# acciones si la condición es verdadera

} vegeta {

# acciones si la condición es falsa

}

**Ejemplo Detallado con Uso de Palabras Claves en Hasklyns:**

Palabra Clave: **mario** y **goku**

* **Inspirado por**: Mario Bros y Dragon Ball Z.
* **Definición**: **mario** se usa para iniciar bucles que repiten acciones un número definido de veces. **goku** se usa para ejecutar un bloque de código si se cumple una condición específica.

**Sintaxis**:

mario (int i = 0; i < 10; i++) {

# acciones a repetir

}

goku (power > 9000) {

# acciones si la condición es verdadera

}

**Cómo usarlo**:

**mario** se utiliza para ciclos determinados por un contador, típicamente para iterar sobre colecciones o repetir acciones. **goku** se utiliza para tomar decisiones en el código basadas en condiciones lógicas o comparaciones.

**Ejemplo en Python**:

for i in range(10):

print(i)

if power > 9000:

print("It's over 9000!")

**Ejemplo en Hasklyns**:

mario (int i = 0; i < 10; i++) {

kamehameha(i);

}

goku (power > 9000) {

kamehameha("It's over 9000!");

}

**Tipos de Datos en Hasklyns**

Definición:

En programación, los tipos de datos son una característica esencial que define el tipo de valor que una variable puede contener. En Hasklyns, los tipos de datos están personalizados con nombres de elementos y personajes de videojuegos para hacer el lenguaje más atractivo y memorable.

**Sintaxis:**

En Hasklyns, la declaración de variables y su tipo sigue una estructura personalizada:

<tipo\_de\_dato\_videojuego> <nombre\_variable> = <valor\_inicial>;

Cada **<tipo\_de\_dato\_videojuego>** refleja un tipo de datos tradicional pero se expresa con un término inspirado en videojuegos.

**Cómo se usa:**

Los tipos de datos en Hasklyns se usan para declarar variables y definir qué tipo de operaciones se pueden realizar con ellas. Estos tipos ayudan a prevenir errores de tipo y aseguran que las funciones del programa se ejecuten correctamente.

Ejemplos en Python y Hasklyns:

**Python:**

int health = 100

string player\_name = "Nathan Drake"

bool is\_alive = True

**Hasklyns (usando videojuegos para inspiración):**

**Mario Bros (para enteros):**  
monedas health = 100; // 'monedas' representa un tipo entero.

**Uncharted (para cadenas):**

aventurero player\_name = "Nathan Drake"; // 'aventurero' para cadenas de texto.  
**The Last of Us (para booleanos):**

superviviente is\_alive = verdadero; // 'superviviente' para booleanos, traducción directa de 'true'.

**Ejemplo Completo en Hasklyns con Uso de Tipos de Datos:**

Palabra Clave: monedas, aventurero, superviviente

Inspirado por: Mario Bros, Uncharted, The Last of Us.

Definición: monedas se usa para enteros, típicamente representando cantidades numéricas. aventurero se usa para cadenas de caracteres, ideal para nombres y textos descriptivos. superviviente se usa para valores booleanos, representando estados de verdadero o falso.

**Sintaxis:**

monedas vida = 100;

aventurero nombre = "Link";

superviviente estaVivo = verdadero;

**Cómo usarlo:**

monedas se utiliza para almacenar y manipular datos numéricos como la salud o el dinero. aventurero es útil para etiquetas y nombres en interfaces o diálogos. superviviente es ideal para controlar lógica de juego, como verificar si un personaje está vivo o activar eventos.

**Ejemplo en Python:**

int health = 100

string name = "Link"

bool isAlive = True

**Ejemplo en Hasklyns**:  
monedas vida = 100;

aventurero nombre = "Link";

superviviente estaVivo = verdadero;

**Operadores en Hasklyns**

**1. Operadores de Asignación**

* **Definición: Los operadores de asignación se utilizan para asignar valores a las variables. En la mayoría de los lenguajes de programación, el operador básico de asignación es =, que asigna el valor del lado derecho al operando del lado izquierdo.**
* **Sintaxis: <variable> = <expresión>;**
* **Cómo usarlo: Se utiliza para inicializar o actualizar el valor de una variable.**

**Ejemplo en Python:**

health = 100  
**Ejemplo en Hasklyns:**

vida = 100;

**Operadores Aritméticos**

* **Definición:** Los operadores aritméticos realizan operaciones matemáticas básicas como suma, resta, multiplicación y división.
* **Sintaxis:**
  + Suma: **<expresión> + <expresión>**
  + Resta: **<expresión> - <expresión>**
  + Multiplicación: **<expresión> \* <expresión>**
  + División: **<expresión> / <expresión>**
* **Cómo usarlo:** Se utilizan para calcular valores numéricos y son fundamentales para cualquier cálculo matemático en la programación.

**Ejemplo en Python:**

result = 10 + 5

**Ejemplo en Hasklyns:**

resultado = 10 + 5;

**Operadores de Igualdad y Relacionales**

* **Definición:** Estos operadores comparan dos valores y devuelven un valor booleano basado en si la comparación es verdadera o falsa.
* **Sintaxis:**
  + Igualdad: **<expresión> == <expresión>**
  + Desigualdad: **<expresión> != <expresión>**
  + Mayor que: **<expresión> > <expresión>**
  + Menor que: **<expresión> < <expresión>**
  + Mayor o igual que: **<expresión> >= <expresión>**
  + Menor o igual que: **<expresión> <= <expresión>**
* **Cómo usarlo:** Utilizados para control de flujo y decisiones lógicas dentro del programa.

**Ejemplo en Python:**

if health >= 100:

print("Full health")

**Ejemplo en Hasklyns:**

si vida >= 100 {

kamehameha("Salud completa");

}

**Sentencias de Control en Hasklyns**

Definición:

Las sentencias de control son estructuras en programación que alteran el flujo de ejecución del código basándose en condiciones dadas o permiten la selección entre múltiples flujos de ejecución. Son cruciales para implementar la lógica de decisión en los programas.

Sintaxis y Uso:

1. **Condicional if y else:**
   * **Hasklyns:** Utiliza nombres de personajes que representan fuerza o decisión.
   * **Sintaxis:**

<nombre\_personaje>(condición) {

// Código a ejecutar si la condición es verdadera

} <nombre\_personaje\_secundario> {

// Código a ejecutar si la condición es falsa

}

* + **Uso:** Usado para ejecutar bloques de código dependiendo de si una condición específica es verdadera o falsa.

1. **Estructura de control switch:**
   * **Hasklyns:** Emplea términos que implican selección o cambio.
   * **Sintaxis:**

switch(<expresión>) {

case <valor1>:

// Código a ejecutar para valor1

break;

case <valor2>:

// Código a ejecutar para valor2

break;

default:

// Código a ejecutar si no se cumplen los casos anteriores

}

* + **Uso:** Permite la ejecución de diferentes partes de código basadas en el valor de una variable. Útil para múltiples rutas de decisión basadas en una sola variable.

Ejemplos en Python y Hasklyns:

**Python (If-Else):**

if health > 50:

print("Keep fighting!")

else:

print("Need healing!")

**Hasklyns (God of War - Kratos para "if", Atreus para "else"):**

kratos (salud > 50) {

kamehameha("¡Continúa la lucha!");

} atreus {

kamehameha("Necesitas curarte");

}

**Python (Switch):**

weapon = "sword"

if weapon == "sword":

print("Sword selected")

elif weapon == "bow":

print("Bow selected")

else:

print("No weapon selected")

**Hasklyns (World of Warcraft - "switch" usando el concepto de selección de arsenal):**

seleccionar(arma) {

caso "espada":

kamehameha("Espada seleccionada");

romper;

caso "arco":

kamehameha("Arco seleccionado");

romper;

porDefecto:

kamehameha("Ninguna arma seleccionada");

}  
**Ciclos en Hasklyns**

Definición:

Los ciclos o bucles permiten repetir una sección de código múltiples veces, ya sea mientras se cumpla una condición (**while**, **do while**) o un número determinado de veces (**for**). Son fundamentales en cualquier lenguaje de programación para realizar tareas repetitivas de manera eficiente.

Sintaxis y Uso:

1. **Ciclo while:**
   * **Hasklyns:** Utiliza personajes conocidos por su resistencia o persistencia.
   * **Sintaxis:**

<nombre\_personaje>(condición) {

// Código a repetir mientras la condición sea verdadera

}

* + **Uso:** Utilizado cuando se necesita repetir un bloque de código mientras una condición específica sea verdadera.

1. **Ciclo do while:**
   * **Hasklyns:** Emplea personajes que simbolizan un esfuerzo continuo, incluso después de un evento.
   * **Sintaxis:**

hacer {

// Código a ejecutar al menos una vez y repetir mientras la condición sea verdadera

} mientras(<nombre\_personaje>(condición));

* + **Uso:** Adecuado cuando el bloque de código necesita ejecutarse al menos una vez antes de verificar la condición.

1. **Ciclo for:**
   * **Hasklyns:** Inspirado en personajes que emprenden viajes o misiones con etapas claras.
   * **Sintaxis:**

<nombre\_personaje>(inicialización; condición; incremento) {

// Código a repetir mientras la condición se cumpla

}

* + **Uso:** Ideal para ciclos donde el número de iteraciones es conocido de antemano.

Ejemplos en Python y Hasklyns:

**Python (While Loop):**

while health > 0:

fight()

**Hasklyns (God of War - "** thor **" para "while"):**

thor (salud > 0) {

combatir();

}

**Python (Do While Loop):** *Notar que Python no tiene una estructura de* ***do while*** *nativa, por lo que se simula con un ciclo* ***while****.*  
  
while True:

fight()

if not condition:

break

**Hasklyns (Naruto - "Naruto" para "do while"):**

hacer {

combatir();

} mientras(naruto(condición));  
**Python (For Loop):**

for i in range(10):

print(i)  
**Hasklyns (Mario Bros - "Mario" para "for"):**

mario (int i = 0; i < 10; i++) {

kamehameha(i);

}

**Modificadores de Acceso en Hasklyns**

Definición:

Los modificadores de acceso determinan el nivel de acceso que tienen otros elementos del programa a las variables y métodos dentro de una clase. En la mayoría de los lenguajes de programación orientados a objetos, los principales modificadores de acceso son **public** (público) y **private** (privado).

1. **Public:**
   * **Hasklyns:** Utiliza un término que sugiera accesibilidad y apertura.
   * **Sintaxis:**

abierto <tipo> <nombre\_variable>;

* + **Uso:** Utilizado para declarar miembros de una clase que pueden ser accesados desde cualquier otra parte del programa.

1. **Private:**
   * **Hasklyns:** Emplea un término que implique restricción o limitación.
   * **Sintaxis:**

oculto <tipo> <nombre\_variable>;

* + **Uso:** Adecuado para miembros de una clase que sólo deben ser accesibles dentro de la misma clase.

Ejemplos en Python y Hasklyns:

**Python (Public y Private):** *Notar que Python no usa palabras clave* ***public*** *o* ***private*** *directamente. Usa* ***\_*** *(underscore) para denotar un convenio de privacidad.*

class Character:

def \_\_init\_\_(self):

self.name = "Joel" # Public por defecto

self.\_secret = "Cannot tell" # Private por convención

# Acceso desde fuera de la clase

character = Character()

print(character.name)

print(character.\_secret) # Accesible pero no recomendado

**Hasklyns (The Last of Us para "public", Grand Theft Auto IV para "private"):**

class Survivor {

abierto relic name = "Joel"; // 'abierto' indica que 'name' es accesible públicamente

oculto secret \_secret = "Cannot tell"; // 'oculto' indica que '\_secret' es privado y sólo accesible dentro de la clase

// Constructor y métodos de la clase aquí

}

// Acceso desde fuera de la clase

Superviviente character = new Survivor();

kamehameha(character.name);

// kamehameha(character.\_secret); Esto resultaría en un error ya que '\_secret' es privado  
**Detalles Específicos por Juego:**

* **The Last of Us (para "public"):**
  + **Palabra Clave:** **abierto**
  + **Ejemplo:**

abierto reliquia nombre = "Ellie";

**Grand Theft Auto IV (para "private"):**

* **Palabra Clave:** **oculto**
* **Ejemplo:**

oculto secreto detalles = "Plan de robo";

Definición:

* **Clases:** Son plantillas para crear objetos que encapsulan datos y funciones relacionadas.
* **Atributos:** Son variables definidas dentro de una clase que almacenan datos sobre el objeto.
* **Constructores:** Son métodos especiales dentro de una clase que se llaman automáticamente al crear una instancia de esa clase. Se utilizan para inicializar los atributos del objeto.
* **Instancias:** Son objetos individuales creados a partir de una clase, cada uno con sus propios datos de atributos.

Sintaxis y Uso:

1. **Clases:**
   * **Hasklyns:** Inspiradas en videojuegos, las clases pueden llevar nombres de lugares o elementos clave dentro del juego.
   * **Sintaxis:**

clase <nombre\_clase> {

// Atributos y métodos

}

* + **Uso:** Se utiliza para definir la estructura y comportamiento de objetos similares dentro de un programa.

1. **Atributos:**
   * **Sintaxis:**

<tipo> <nombre\_atributo>;

* + **Uso:** Se utilizan para almacenar información que pertenece a los objetos.

1. **Constructores:**
   * **Sintaxis:**

constructor(<parámetros>) {

// Inicialización de atributos

}

* + **Uso:** Inicializa los atributos de una nueva instancia de la clase.

1. **Instancias:**
   * **Sintaxis:**

<nombre\_clase> <nombre\_instancia> = nuevo <nombre\_clase>(<argumentos>);

* + **Uso:** Crea un objeto individual a partir de la clase definida.

Ejemplos en Python y Hasklyns:

**Python (Clases y Atributos):**

class Hero:

def \_\_init\_\_(self, name, strength):

self.name = name

self.strength = strength

# Creación de una instancia

link = Hero("Link", 100)

**Hasklyns (Warcraft - utilizando "Azeroth" para "Hero"):**

clase Azeroth {

reliquia nombre;

fuerza poder;

constructor(reliquia nombreHeroe, fuerza poderHeroe) {

nombre = nombreHeroe;

poder = poderHeroe;

}

}

// Creación de una instancia

Azeroth arthas = nuevo Azeroth("Arthas", 150);  
**Detalles Específicos por Videojuego:**

* **Warcraft (para "Azeroth" en lugar de "Hero"):**
  + **Definición:** 'Azeroth' como clase, tomando el nombre del mundo de Warcraft, donde se encuentran héroes y villanos.
  + **Ejemplo en Hasklyns:**

clase Azeroth {

reliquia nombre;

fuerza poder;

constructor(reliquia nombreHeroe, fuerza poderHeroe) {

nombre = nombreHeroe;

poder = poderHeroe;

}

}

// Creación de una instancia de Arthas, un famoso personaje de Warcraft

Azeroth arthas = nuevo Azeroth("Arthas", 150);

**Alcances de Variables en Hasklyns**

Definición:

El alcance de una variable define la región del código donde una variable es accesible. En programación, los alcances típicos son local, donde la variable es accesible sólo dentro de un bloque de código o función específica, y global, donde la variable es accesible desde cualquier parte del programa.

Sintaxis y Uso:

1. **Variables Locales:**
   * **Hasklyns:** Utilizamos términos que sugieren una limitación o un contexto específico.
   * **Sintaxis:**

// Dentro de una función o bloque de código

<tipo> <nombre\_variable> = <valor\_inicial>;

* + **Uso:** Usadas para operaciones dentro de una función o bloque específico, no accesibles fuera de este.

1. **Variables Globales:**
   * **Hasklyns:** Empleamos palabras que indican amplitud o universalidad.
   * **Sintaxis:**

global <tipo> <nombre\_variable> = <valor\_inicial>;

* + **Uso:** Declaradas fuera de cualquier función o bloque, accesibles desde cualquier parte del programa.

Ejemplos en Python y Hasklyns:

**Python (Variables Locales y Globales):**

global\_var = "Available everywhere"

def function():

local\_var = "Available only here"

print(global\_var)

print(local\_var)

function()

print(global\_var)

# print(local\_var) # This would raise an error because local\_var is not accessible here.

**Hasklyns (Usando el universo de Warcraft y Grand Theft Auto IV):**

global reliquia mundo = "Azeroth"; // 'reliquia' para una variable global del tipo string

funcion batalla() {

local arma espada = "Excalibur"; // 'arma' como tipo local para una variable dentro de una función

kamehameha (mundo); // Accede a la variable global

kamehameha (espada); // Accede a la variable local

}

batalla();

kamehameha (mundo); // Funciona bien

// kamehameha (espada); // Esto generaría un error, 'espada' no es accesible aquí.

**Detalles Específicos por Videojuego:**

* **Warcraft (para "global" y "local"):**
  + **Definición de 'global':** 'reliquia' para indicar variables que son tesoros accesibles universalmente.
  + **Definición de 'local':** 'arma' para denotar objetos que solo son útiles en contextos específicos de batalla.
  + **Ejemplo en Hasklyns:**

global reliquia mundo = "Azeroth"; // Accesible universalmente

funcion batalla() {

local arma espada = "Excalibur"; // Solo accesible dentro de 'batalla'

anunciar(mundo);

anunciar(espada);

}

**Métodos y Parámetros en Hasklyns**

Definición:

* **Métodos:** Funciones definidas dentro de una clase que describen las acciones o comportamientos que los objetos de esa clase pueden realizar.
* **Parámetros:** Son variables especificadas en la definición de un método que aceptan valores cuando el método es llamado. Los parámetros permiten que los métodos actúen en datos específicos y sean más flexibles y reutilizables.

Sintaxis y Uso:

1. **Definición de Métodos:**
   * **Hasklyns:** Emplea nombres relacionados con acciones o habilidades de personajes de videojuegos.
   * **Sintaxis:**

funcion <nombre\_metodo>(<tipo> <parametro1>, <tipo> <parametro2>, ...) {

// Código del método

}

* + **Uso:** Se utilizan para encapsular lógicas que operan sobre los datos de los objetos, ofreciendo funcionalidades específicas como calcular valores, modificar atributos o interactuar con otros objetos.

1. **Parámetros de Métodos:**
   * **Sintaxis:**

<tipo> <nombre\_parametro>

* + **Uso:** Permiten que los métodos reciban datos externos o configuraciones cuando son invocados, haciendo los métodos más dinámicos y versátiles.

Ejemplos en Python y Hasklyns:

**Python (Métodos y Parámetros):**

class Player:

def \_\_init\_\_(self, name, level):

self.name = name

self.level = level

def attack(self, target):

print(f"{self.name} attacks {target} at level {self.level}")

**Hasklyns (Usando personajes de Warcraft y Grand Theft Auto IV):**

clase Jugador {

reliquia nombre;

nivel poder;

constructor(reliquia nombreJugador, nivel nivelJugador) {

nombre = nombreJugador;

poder = nivelJugador;

}

accion atacar(reliquia objetivo) {

kamehameha (nombre + " ataca a " + objetivo + " con poder de " + poder);

}

}

**Detalles Específicos por Videojuego:**

* **Warcraft (para "acción"):**
  + **Definición:** 'acción' para definir métodos que involucran ejecución de habilidades o poderes.
  + **Ejemplo en Hasklyns:**

accion atacar(reliquia objetivo) {

anunciar(nombre + " ataca a " + objetivo + " con poder de " + poder);

}  
**Grand Theft Auto IV (para "constructor"):**

* **Definición:** 'constructor' para definir el método especial que configura nuevas instancias de la clase.
* **Ejemplo en Hasklyns:**

constructor(reliquia nombreJugador, nivel nivelJugador) {

nombre = nombreJugador;

poder = nivelJugador;

}

**Impresión de Información en Hasklyns**

Definición:

La impresión de información es una operación común en programación que permite mostrar mensajes en la consola del usuario o en una ventana emergente, facilitando la interacción directa o el debug del programa.

Sintaxis y Uso:

* **Hasklyns:**
  + **Palabra clave:** **kamehameha** - en referencia al poderoso ataque de energía en Dragon Ball Z, utilizado aquí para "lanzar" información a la consola.
  + **Sintaxis:**

kamehameha(<mensaje>);

* + **Uso:** Utilizado para emitir mensajes de texto, valores de variables, o resultados de funciones directamente a la consola, ayudando en la depuración y seguimiento del flujo del programa.

Ejemplos en Python y Hasklyns:

**Python (Impresión en Consola):**

print("Hello, world!")

**Hasklyns (Usando la palabra "kamehameha"):**

kamehameha("¡Hola, mundo!");

**Ejemplo Completo en Hasklyns:**

Para proporcionar un contexto más detallado, imagine que estamos trabajando en una función que calcula el poder de un personaje basado en su nivel y experiencia, y queremos imprimir el resultado:

**Python:**

def calculate\_power(level, experience):

power = level \* experience

print("Power level:", power)

return power

calculate\_power(5, 100)

**Hasklyns (Usando "Kamehameha" y Dragon Ball Z para el contexto):**

funcion calcularPoder(nivel, experiencia) {

poder = nivel \* experiencia

kamehameha("Nivel de poder: " + poder);

return poder

}

calcularPoder(5, 100);

**Enumeraciones en Hasklyns**

Definición:

Una enumeración es un tipo de datos especial que consiste en un conjunto de constantes nombradas. En la programación, las enumeraciones son utilizadas para definir variables que pueden tener un conjunto restringido de valores posibles, lo que ayuda a mejorar la legibilidad del código y a reducir errores relacionados con valores inválidos.

Sintaxis y Uso:

* **Hasklyns:**
  + **Sintaxis:**

enumeracion <nombre\_enumeracion> {

<opcion1>,

<opcion2>,

<opcion3>,

...

}

* + **Uso:** Las enumeraciones se utilizan para definir un tipo que puede tomar uno de los varios posibles valores predefinidos. Son útiles para manejar opciones fijas, como estados, configuraciones, tipos de ítems en un juego, etc.

Ejemplos en Python y Hasklyns:

**Python (Enumeraciones):**

from enum import Enum

class WeaponType(Enum):

SWORD = 1

SHIELD = 2

BOW = 3

# Uso de la enumeración

weapon = WeaponType.SWORD

print("Selected weapon:", weapon.name)

**Hasklyns (Usando el universo de "The Elder Scrolls" para la inspiración):**

enumeracion TipoArma {

ESPADA,

ESCUDO,

ARCO

}

// Creación de una variable de tipo enumerado

TipoArma arma = TipoArma.ESPADA;

anunciar("Arma seleccionada: " + arma);

**Ejemplo Detallado en Hasklyns:**

Imaginemos que estamos trabajando en un RPG donde los jugadores pueden elegir entre diferentes tipos de armas, y queremos utilizar enumeraciones para manejar estas opciones.

Contexto de Juego: **"The Elder Scrolls"**

* **Definición de Enumeración:**
  + **Palabra Clave:** **enumeracion**
  + **Ejemplo en Hasklyns:**

enumeracion TipoArma {

ESPADA, // Ideal para combate cuerpo a cuerpo

ESCUDO, // Ofrece defensa adicional

ARCO // Perfecto para ataques a distancia

}

**Uso de la Enumeración:**

* **Cómo usarlo:** Las enumeraciones son ideales para definir tipos que representan colecciones de constantes relacionadas y para asegurar que las variables solo contengan valores válidos dentro de esas constantes.
* **Ejemplo en Hasklyns:**

TipoArma armaSeleccionada = TipoArma.ARCO;

anunciar("Arma seleccionada: " + armaSeleccionada);

**Bloques de Código en Hasklyns**

Definición:

Un bloque de código es una sección de código encerrada entre llaves **{}** que define un ámbito para las variables y agrupa varias instrucciones en una sola unidad de ejecución. Esto es esencial en la estructura de control de flujo y la organización del código.

Sintaxis y Uso:

* **Sintaxis:**

{

// Declaraciones y expresiones que se ejecutan juntas

}

* **Uso:** Los bloques de código se utilizan para:
  + Delimitar los cuerpos de las funciones.
  + Definir el ámbito de ejecución en estructuras de control como condicionales y bucles.
  + Agrupar declaraciones que deben ejecutarse juntas.

Ejemplos en Python y Hasklyns:

**Python:** Python utiliza la indentación para definir bloques de código, que es una característica central de su sintaxis.

def function():

# Bloque de función

if True:

# Bloque condicional

print("Hello, world!")

for i in range(5):

# Bloque de bucle

print(i)

**Hasklyns:** En Hasklyns, los bloques se delimitan con llaves **{}** como en muchos otros lenguajes de programación como C, Java o JavaScript.

funcion() {

// Bloque de función

goku (verdadero) {

// Bloque condicional

kamehameha("¡Hola, mundo!");

}

}

mario (int i = 0; i < 5; i++) {

// Bloque de bucle

kamehameha(i);

}

**Ejemplo Detallado en Hasklyns:**

Supongamos que estamos trabajando en una función en Hasklyns que calcula los resultados de un combate en un juego, utilizando elementos inspirados en diversos videojuegos para estructurar la lógica.

funcion calcularResultadoCombate(heroe vida, enemigo vida) {

// Inicio del bloque de función

si (heroe vida > enemigo vida) {

// Inicio del bloque condicional

kamehameha("El héroe gana el combate!");

} sino {

kamehameha("El enemigo gana el combate!");

}

// Fin del bloque condicional

}

// Fin del bloque de función

**Ejemplo de Codigo HasKlyns**

**// Clase Arsenal para gestionar un inventario de artefactos**

**clase Arsenal {**

**// Variables públicas para almacenar los artefactos y el valor total del inventario**

**abierto diccionario artefactos;**

**abierto entero valorTotal;**

**// Constructor que inicializa el inventario y el valor total a cero**

**constructor() {**

**artefactos = {};**

**valorTotal = 0;**

**}**

**// Método para agregar artefactos al inventario**

**funcion agregarArtefacto(artefacto, cantidad, precio) {**

**// Si el artefacto ya existe, aumenta su cantidad**

**si (artefacto en artefactos) {**

**artefactos[artefacto]['cantidad'] += cantidad;**

**} sino {**

**// Si es un nuevo artefacto, crea una entrada en el diccionario**

**artefactos[artefacto] = {'cantidad': cantidad, 'precio': precio};**

**}**

**// Aumenta el valor total del inventario**

**valorTotal += cantidad \* precio;**

**}**

**// Método para eliminar artefactos del inventario**

**funcion eliminarArtefacto(artefacto, cantidad) {**

**// Verifica que exista el artefacto y que la cantidad a eliminar sea adecuada**

**si (artefacto en artefactos && artefactos[artefacto]['cantidad'] >= cantidad) {**

**artefactos[artefacto]['cantidad'] -= cantidad;**

**valorTotal -= cantidad \* artefactos[artefacto]['precio'];**

**// Si no quedan más artefactos, elimina el artefacto del diccionario**

**si (artefactos[artefacto]['cantidad'] == 0) {**

**eliminar artefactos[artefacto];**

**}**

**}**

**}**

**// Método para mostrar la lista de artefactos y el valor total del inventario**

**funcion mostrarArsenal() {**

**// Itera sobre cada artefacto en el diccionario y muestra sus detalles**

**mario (cada artefacto, detalles en artefactos) {**

**kamehameha(artefacto + ": " + detalles['cantidad'] + " unidades, $" + detalles['precio'] + " cada uno");**

**}**

**kamehameha("Valor Total del Arsenal: $" + valorTotal);**

**}**

**}**

**// Uso de la clase Arsenal para agregar, eliminar y mostrar artefactos**

**arsenal = nuevo Arsenal();**

**arsenal.agregarArtefacto("Espada", 2, 300);**

**arsenal.agregarArtefacto("Escudo", 1, 150);**

**arsenal.eliminarArtefacto("Espada", 1);**

**arsenal.mostrarArsenal();**

**Ejemplo de Codigo Python**

**class Inventory:**

**# Constructor de la clase**

**def \_\_init\_\_(self):**

**self.items = {} # Diccionario para almacenar ítems**

**self.total\_value = 0 # Valor total del inventario**

**# Método para añadir ítems al inventario**

**def add\_item(self, item, quantity, price):**

**if item in self.items:**

**self.items[item]['quantity'] += quantity # Añade a la cantidad existente**

**else:**

**self.items[item] = {'quantity': quantity, 'price': price} # Crea nuevo ítem**

**self.total\_value += quantity \* price # Actualiza el valor total**

**# Método para remover ítems del inventario**

**def remove\_item(self, item, quantity):**

**if item in self.items and self.items[item]['quantity'] >= quantity:**

**self.items[item]['quantity'] -= quantity # Reduce la cantidad**

**self.total\_value -= quantity \* self.items[item]['price'] # Ajusta el valor total**

**if self.items[item]['quantity'] == 0:**

**del self.items[item] # Elimina ítem si la cantidad es 0**

**# Método para mostrar el inventario**

**def display\_inventory(self):**

**for item, details in self.items.items():**

**print(f"{item}: {details['quantity']} units, ${details['price']} each")**

**print(f"Total Inventory Value: ${self.total\_value}")**

**# Creación de la instancia y uso de métodos**

**inventory = Inventory()**

**inventory.add\_item("Sword", 2, 150)**

**inventory.add\_item("Shield", 1, 100)**

**inventory.remove\_item("Sword", 1)**

**inventory.display\_inventory()**

Los códigos tanto en **Python** como en **Hasklyns** son ejemplos de cómo se podría implementar un sistema de inventario básico en un programa. Aquí está la explicación detallada de lo que hace cada parte del código y cómo funcionan en conjunto:

**Funcionalidades del Código**

1. Clase de Inventario (Inventory en Python, Almacen en Hasklyns):
   * Constructor (\_\_init\_\_ en Python, constructor en Hasklyns):
     + Este método inicializa la clase creando un diccionario vacío (items en Python, objetos en Hasklyns) para almacenar los objetos del inventario.
     + Establece el total\_value o valorTotal a cero, que representará el valor monetario total de todos los ítems en el inventario.
2. Método para Agregar Ítems (add\_item en Python, agregarObjeto en Hasklyns):
   * Este método acepta tres parámetros: el nombre del ítem (item o objeto), la cantidad del ítem (quantity o cantidad), y el precio por unidad del ítem (price o precio).
   * Funcionamiento:
     + Si el ítem ya existe en el inventario (verificado por item in self.items en Python y objeto en objetos en Hasklyns), simplemente incrementa la cantidad existente del ítem en el inventario.
     + Si el ítem no existe, crea una nueva entrada en el diccionario del inventario para ese ítem con la cantidad y precio proporcionados.
     + Independientemente de si el ítem es nuevo o ya existente, el método actualiza el total\_value o valorTotal agregando el valor del ítem añadido (cantidad multiplicada por el precio).
3. Método para Mostrar el Inventario (display\_inventory en Python, mostrarAlmacen en Hasklyns):
   * Este método no toma parámetros y se utiliza para imprimir todos los ítems en el inventario junto con sus detalles (cantidad y precio por unidad).
   * Funcionamiento:
     + Itera sobre cada ítem en el diccionario del inventario.
     + Utiliza la función print en Python o kamehameha en Hasklyns para mostrar cada ítem con su cantidad y precio.
     + Finalmente, muestra el valor total del inventario.

Ejecución de Métodos en la Instancia de la Clase

* Se crea una instancia de la clase Inventory o Almacen.
* Se utilizan los métodos de esta instancia para:
  + Agregar varios ítems al inventario, como espadas y escudos, con cantidades y precios específicos.
  + Mostrar todo el inventario después de las adiciones para verificar los detalles y el valor total acumulado.