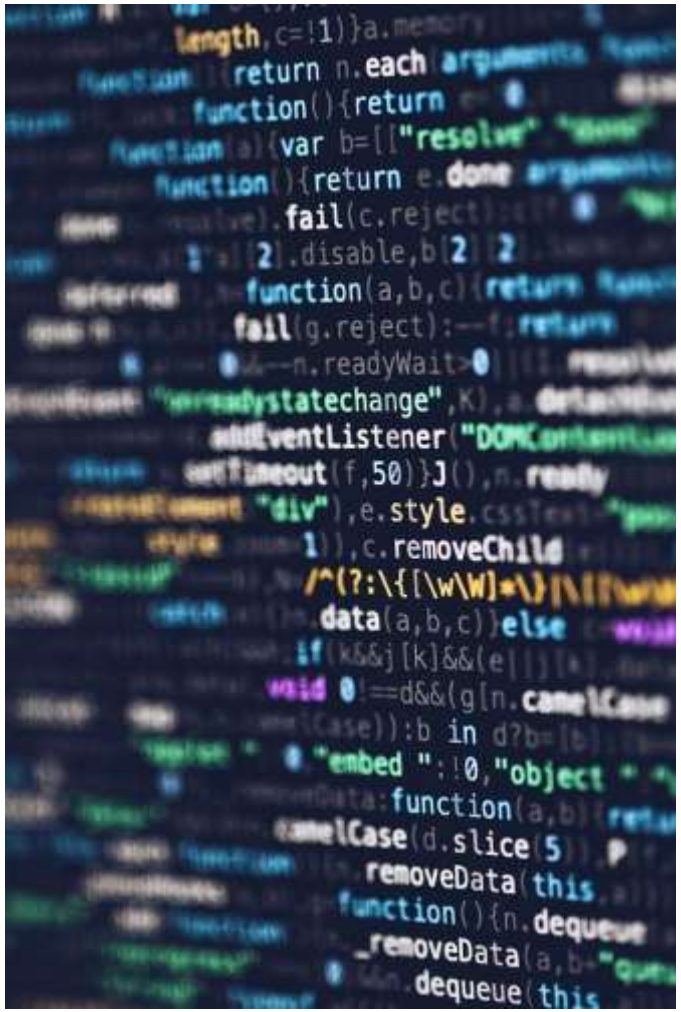
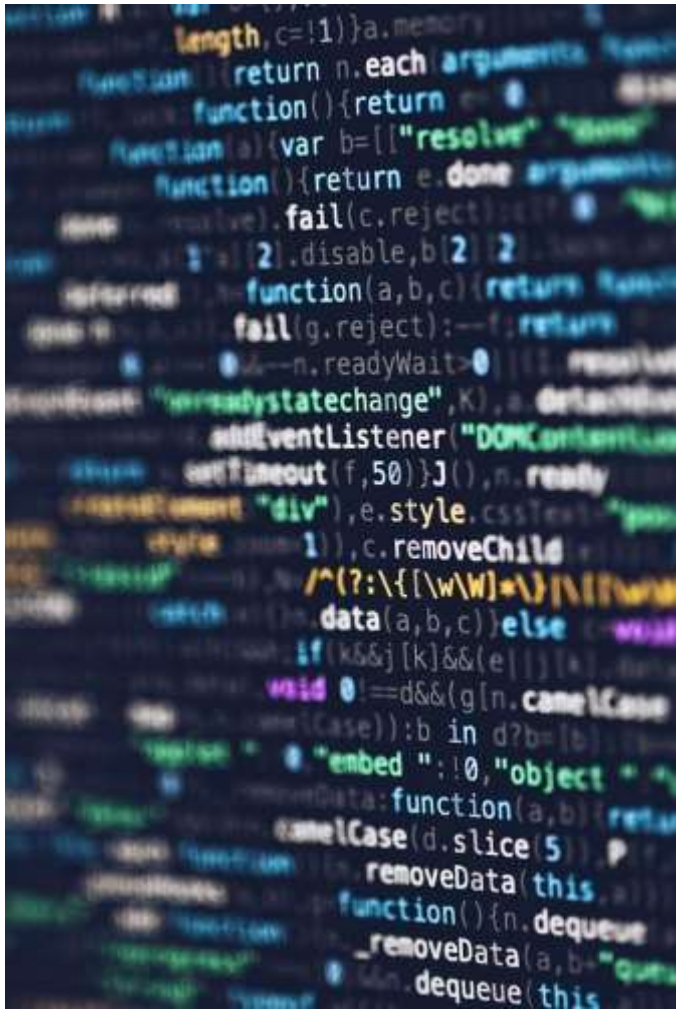
UNIVERSIDAD NACIONAL GUILLERMO BROWN

Conceptos y Paradigmas de Lenguajes de Programación

Clase 2



SINTAXIS y SEMÁNTICA

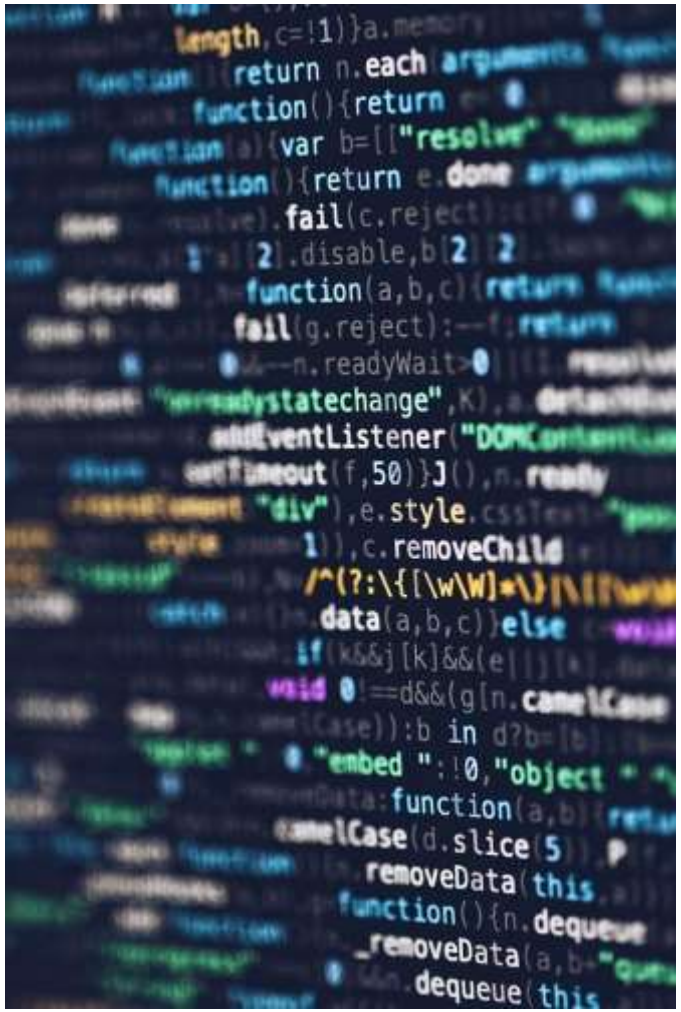


Sintaxis y Semántica

En una definición un poco mas técnica, un lenguaje de programación es una **notación formal** para describir algoritmos a ser ejecutados en una computadora.

sintaxis

Lenguaje de programación

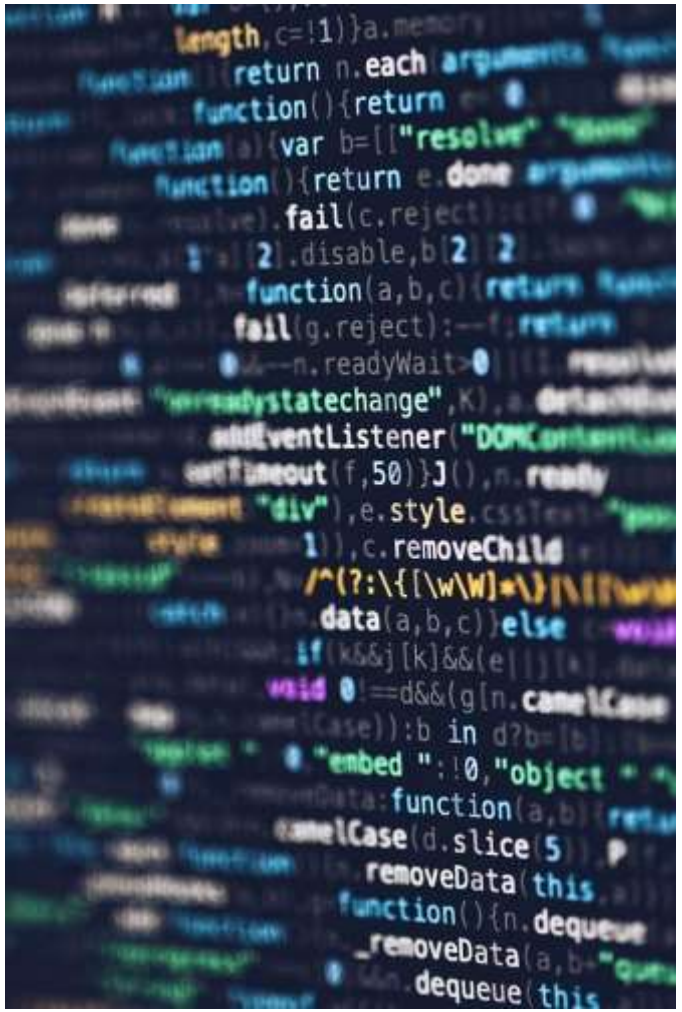


Sintaxis y Semántica

Definiciones

- **Sintaxis**: Conjunto de reglas que definen como componer letras, dígitos y otros caracteres para formar los programas
- **Semántica**: Conjunto de reglas para dar significado a los programas sintácticamente válidos, asigna significado computacional a las cadenas válidas en la sintaxis

v: array [1..10] of integer; -------- en Pascal y int v[10]; ---------- en C



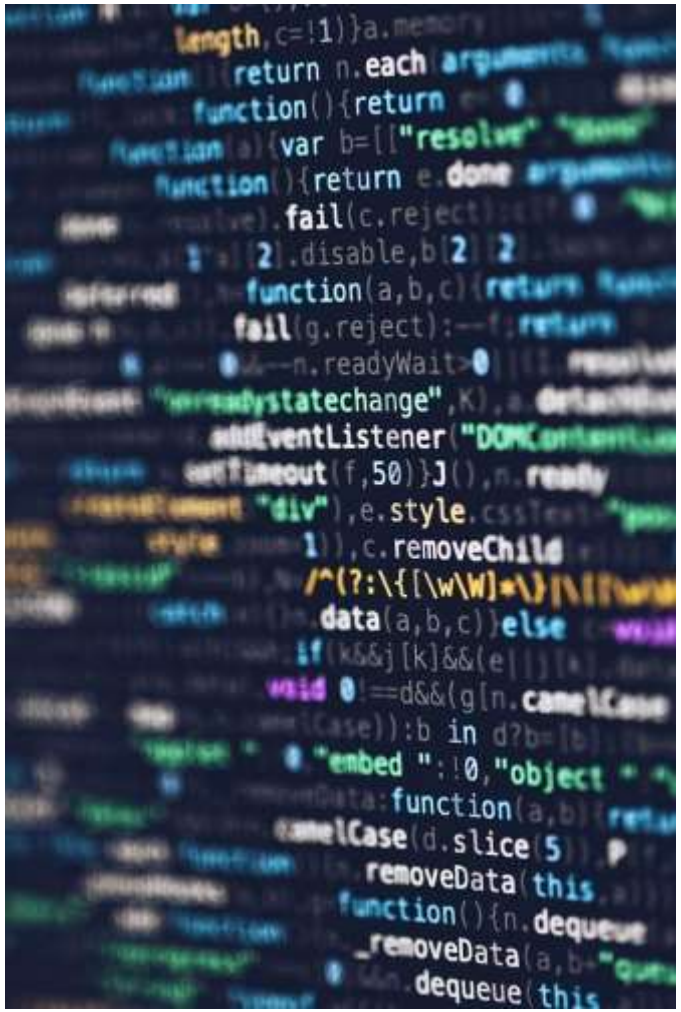
Sintaxis y Semántica

¿Cuál es la utilidad de definir y conocer la sintaxis y la semántica de un lenguaje? ¿Quiénes se benefician?

- Programadores.
- Implementador (Compilador).

Además, La definición de la sintaxis y la semántica de un lenguaje de programación proporcionan mecanismos para que una persona o una computadora pueda decir:

- Si el programa es válido.
- Si es válido, qué significa.

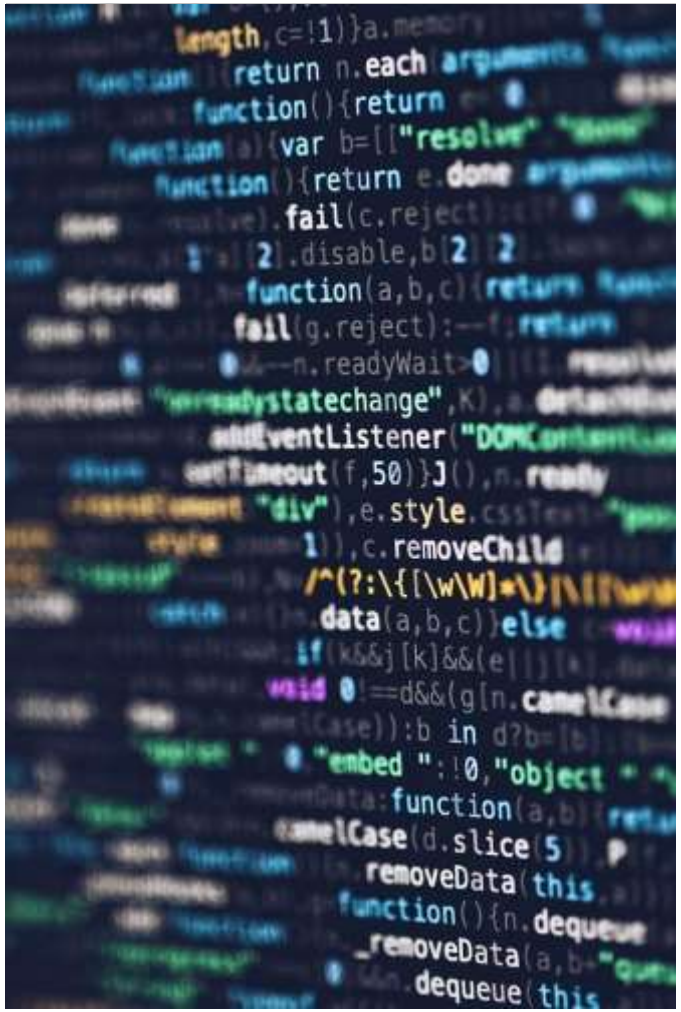


Sintaxis

La sintaxis establece reglas que definen cómo deben combinarse las componentes básicas, para formar sentencias y programas.

Elementos de la sintaxis :

- Alfabeto o conjunto de caracteres
- identificadores
- Operadores
- Palabra clave y palabra reservada
- Comentarios y uso de blancos



Características de la sintaxis

✓ **Palabra clave o keywords**, son palabras claves que tienen un significado dentro de un contexto.

✓ **Palabra reservada**, son palabras claves que además no pueden ser usadas por el programador como identificador de otra entidad.

Ventajas de su uso:

- Permiten al compilador y al programador expresarse claramente
- Hacen los programas más legibles y permiten una rápida traducción

Soluciones para evitar confusión entre palabras claves e identificadores :

- Usar palabras reservadas
- Identificarlas de alguna manera (Ej. Algol) usa ‘PROGRAM ‘END
- Libre uso y determinar de acuerdo al contexto. Ej: if if=1 then if=0;

Ejemplos de lenguajes con uso de palabras reservadas:

•C ej.:auto,break ,case ,char ,const ,continue ,default ,do ,double

•Pascal ej.: absolute, and, array, begin, const, div, do, downto, else, if, in,label,mod,not,of, packed, procedure, record, set, shr, then,to, unit, uses, var, while, xor, etc

Sintaxis

Estructura sintáctica :

■ Vocabulario

Conjunto de caracteres y palabras necesarias para construir expresiones, sentencias y programas.

Ej: identificadores, operadores, palabras claves, etc. Las words no son elementales se construyen a partir del alfabeto

■ Expresiones

+ Son funciones que a partir de un conjunto de datos devuelven un resultado.

+ Son bloques sintácticos básicos a partir de los cuales se construyen las sentencias y programas

■ Sentencias

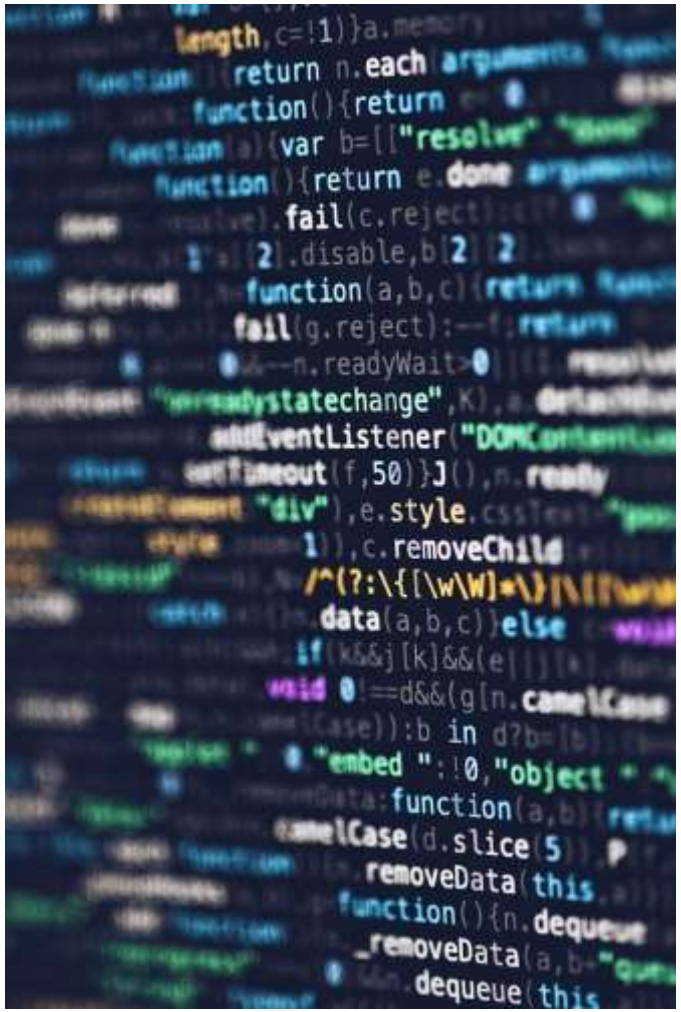
+ Componente sintáctico más importante.

+ Tiene un fuerte impacto en la facilidad de escritura y legibilidad

+ Hay sentencias simples, estructuradas y anidadas.

Importancia de la sintaxis

- La sintaxis es importante porque permite escribir código que funcione correctamente
- La sintaxis facilita el aprendizaje y la aplicación de un lenguaje de programación



Sintaxis

Reglas Léxicas y Sintácticas

■ Reglas léxicas: Conjunto de reglas para formar las “word”, a partir de los caracteres del alfabeto

ejemplo :

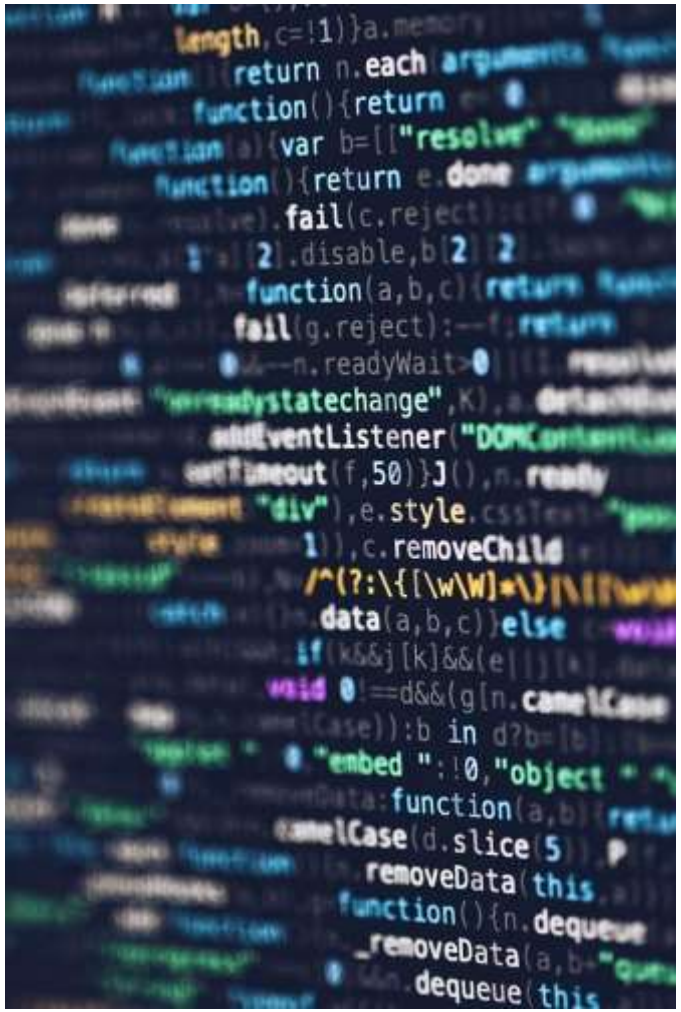
Diferencias entre mayúsculas y minúsculas

•Símbolo de distinto.

En C != en Pascal <>

■ Reglas sintácticas: Conjunto de reglas que definen como formar las “expresiones” y “sentencias”

•El If en C no lleva “”then””, en Pascal si



Sintaxis

Tipos de Sintaxis

- **ABSTRACTA**

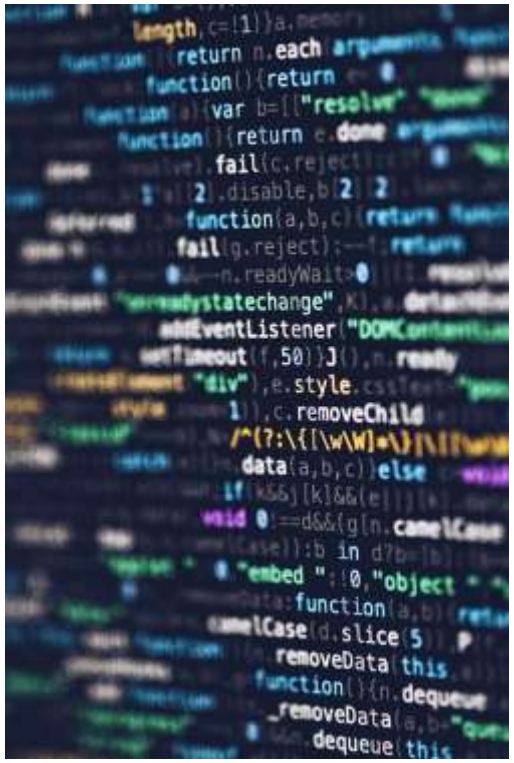
  Se refiere básicamente a la estructura

- **CONCRETA**

  Se refiere básicamente a la parte léxica

- **PRAGMÁTICA**

  Se refiere básicamente al uso práctico



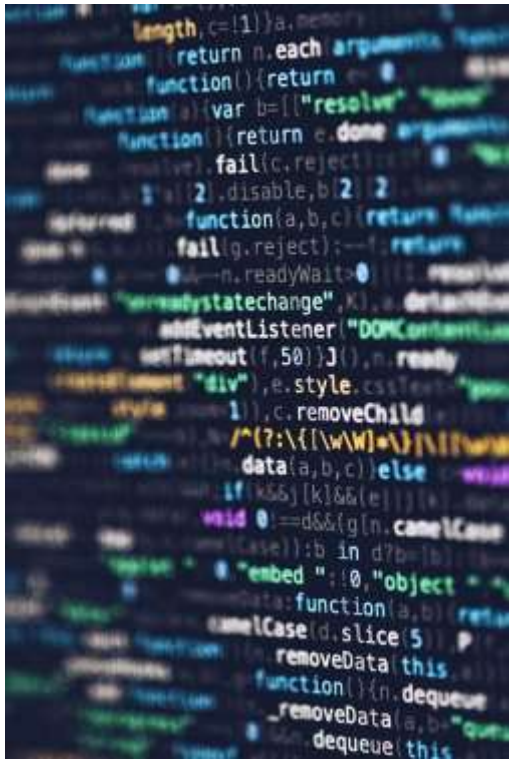
Ejemplo de sintaxis concreta y abstracta

En C
while (x!= y)
{ → Forma de encerrar un bloque

};

En Pascal
while x<>y do
Begin → Forma de encerrar un bloque
---
end

Las expresiones anteriores, son diferentes respecto a la sintaxis concreta, porque existen diferencias léxicas entre ellas ,pero son iguales respecto a la sintaxis abstracta, ya que ambas tienen la misma estructura
while condición bloque



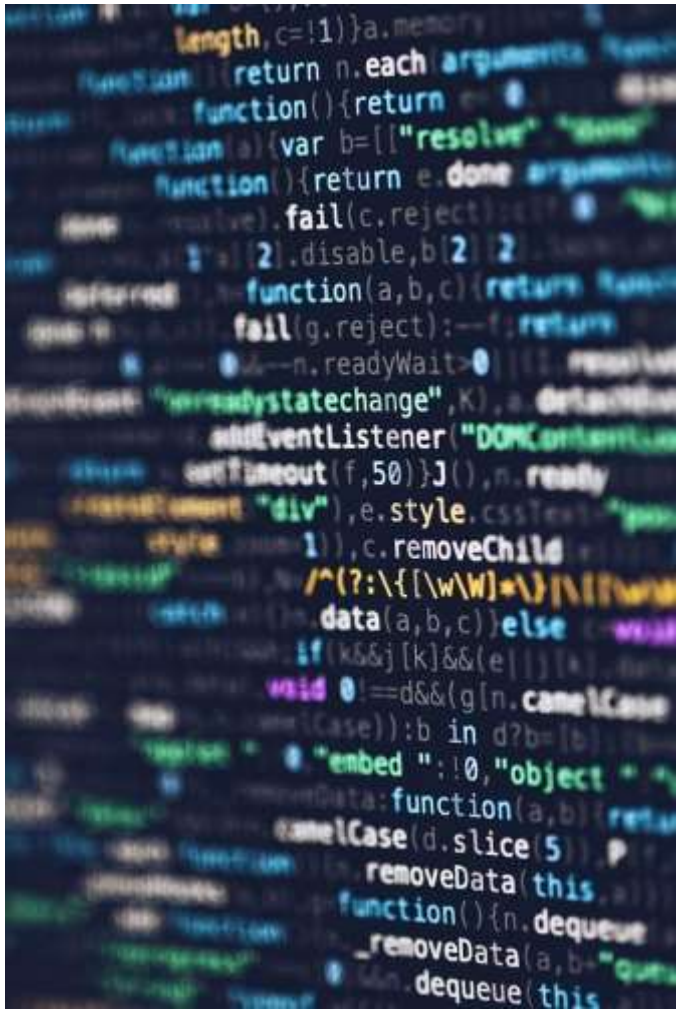
Ejemplo de sintaxis Pragmática

Ej1.
<> es mas legible que !=

Ej2.
En C y Pascal {} o begin-end pueden omitirse si el bloque esta compuesto por una sola sentencia

while (x!=y) x=y+1

Pragmáticamente puede conducir a error ya que si se necesitara agregar una sentencia debe agregarse el begin end o las {}.



Sintaxis

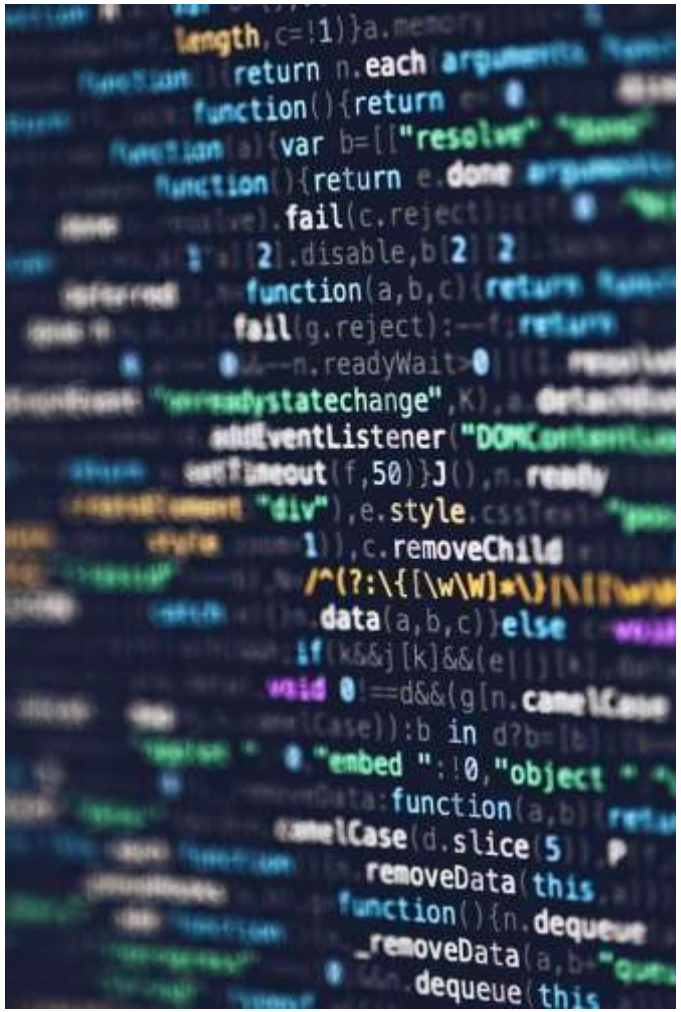
Ejemplo de sintaxis pragmática:

Ej1. <> es mas legible que !=

Ej2. En C y Pascal {} o begin-end pueden omitirse si el bloque esta compuesto por una sola sentencia

```
while (x!=y) x=y+1
```

Pragmáticamente puede conducir a error ya que si se necesitara agregar una sentencia debe agregarse el begin end o las {}.



Sintaxis

Como definir la sintaxis:

■ Se necesita una descripción finita para definir un conjunto infinito (conjunto de todos los programas bien escritos)

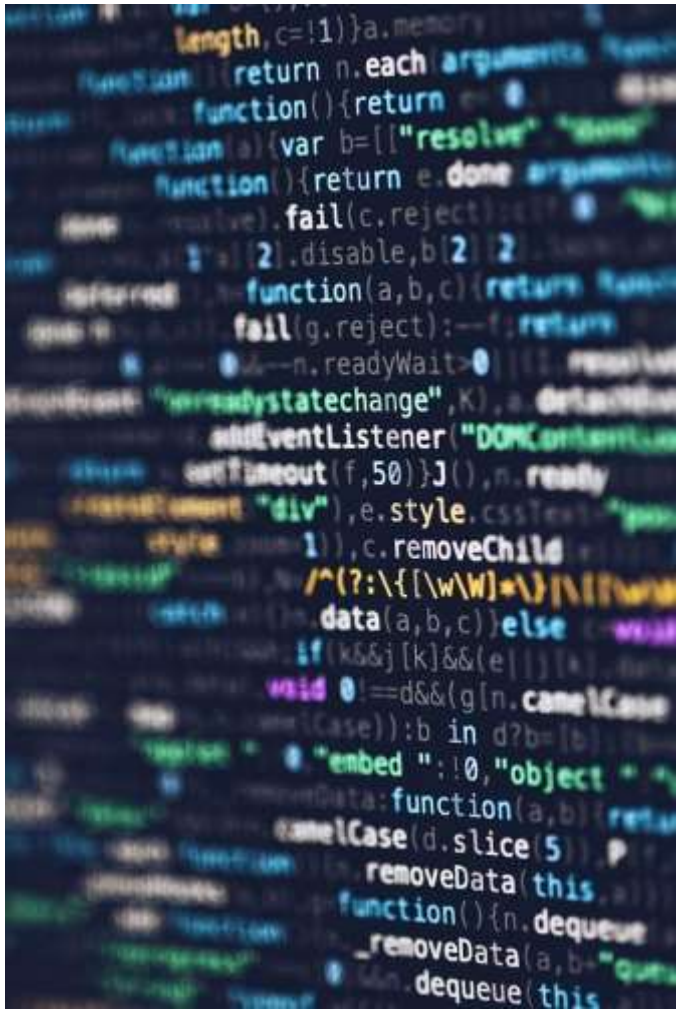
■ Formas para definir la sintaxis:

+ Lenguaje natural. Ej.: Fortran

+ Utilizando la gramática libre de contexto, definida por Backus y Naun:

BNF. Ej: Algol

+ Diagramas sintácticos son equivalentes a BNF pero mucho mas intuitivos

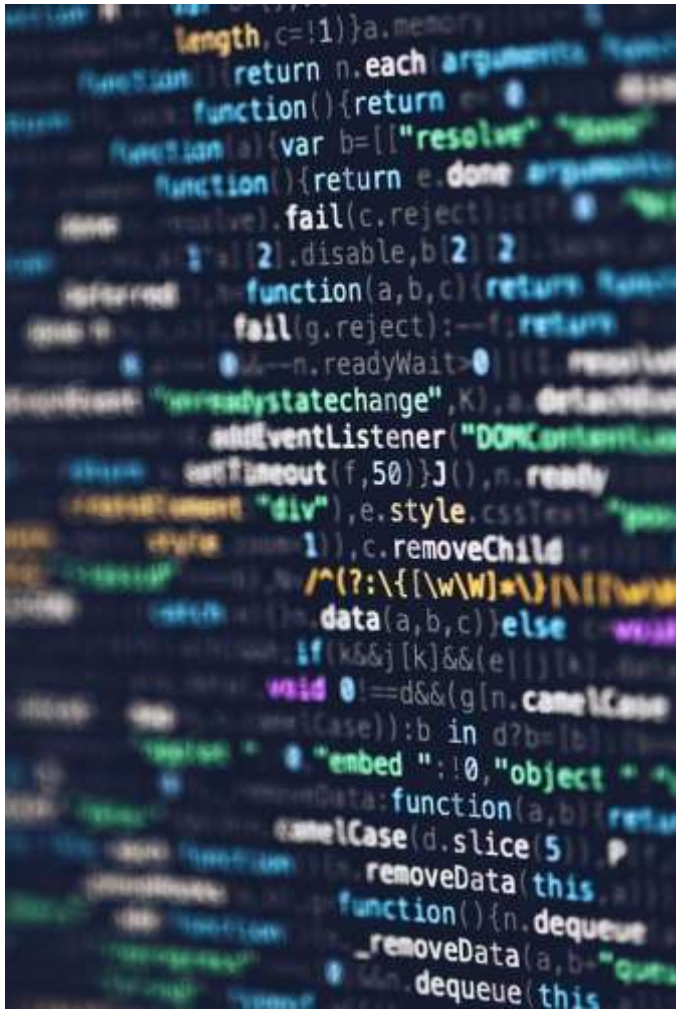


Sintaxis - Errores de sintaxis

Los errores de sintaxis son los más comunes y pueden ser fáciles de detectar.

Sin embargo, pueden ser difíciles de depurar si no se tiene experiencia en programación.Los errores de sintaxis pueden incluir:

- **Sintaxis incorrecta de instrucciones**: La sintaxis incorrecta de instrucciones puede llevar a errores de compilación o ejecución.
  Por ejemplo, un punto y coma (;) faltante al final de una instrucción puede generar un error.
- **Identificadores no válidos**: Los identificadores no válidos pueden incluir palabras reservadas, caracteres especiales o espacios en blanco no permitidos en el nombre de una variable o función.
- **Faltan paréntesis o corchetes**: Los paréntesis y corchetes se utilizan para agrupar expresiones y definir bloques de código.
  La falta de paréntesis o corchetes puede generar errores de sintaxis.

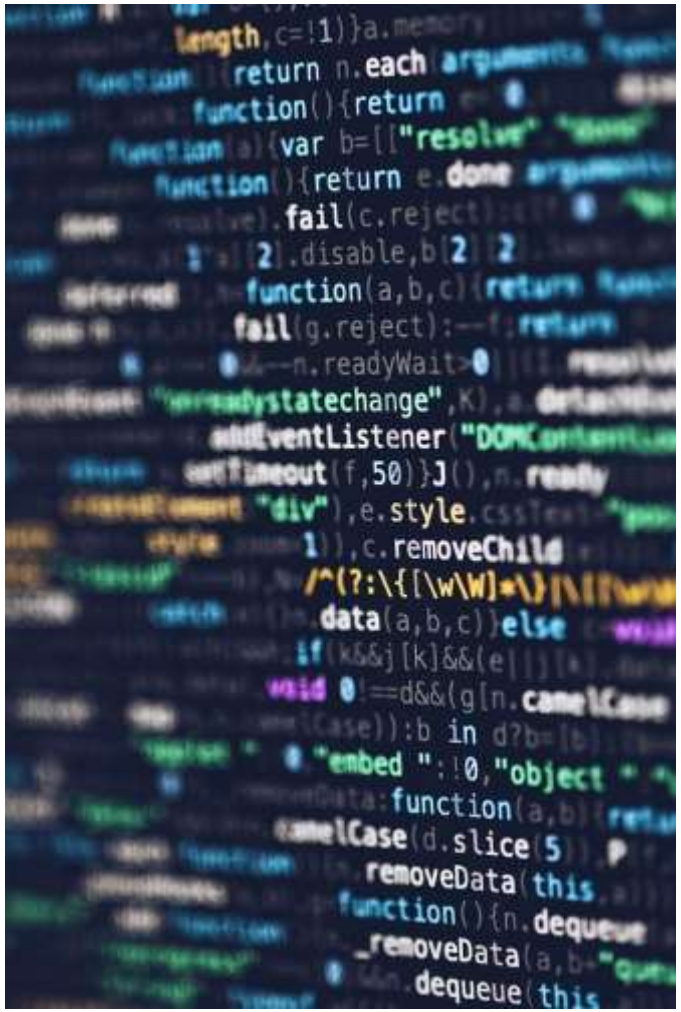


Semántica

Definiciones

Cuando hablamos de programación, a menudo nos centramos en la sintaxis, es decir, la forma en que escribimos nuestras instrucciones en un lenguaje de programación. Sin embargo, hay algo igualmente importante, si no más, que a menudo pasamos por alto: la **semántica**.

*La semántica es el **significado** detrás de las instrucciones que creamos y es lo que la computadora interpreta cuando ejecuta nuestro código.*



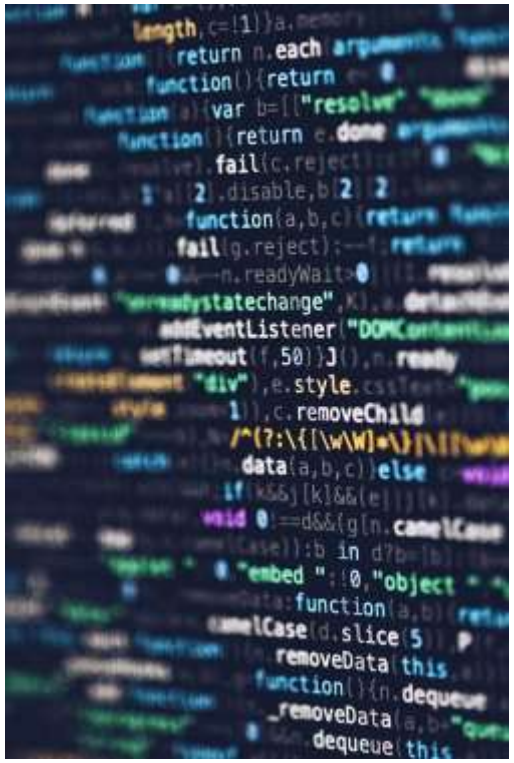
Semántica

Un Ejemplo

vamos a imaginar que estamos escribiendo un programa para servir una taza de café. La sintaxis sería cómo organizas las instrucciones, como “pon agua en la cafetera” y “agrega café molido”. La semántica sería lo que esas instrucciones realmente significan: llenar la cafetera con una cantidad específica de agua y agregar una cantidad precisa de café molido.

Además la semántica:

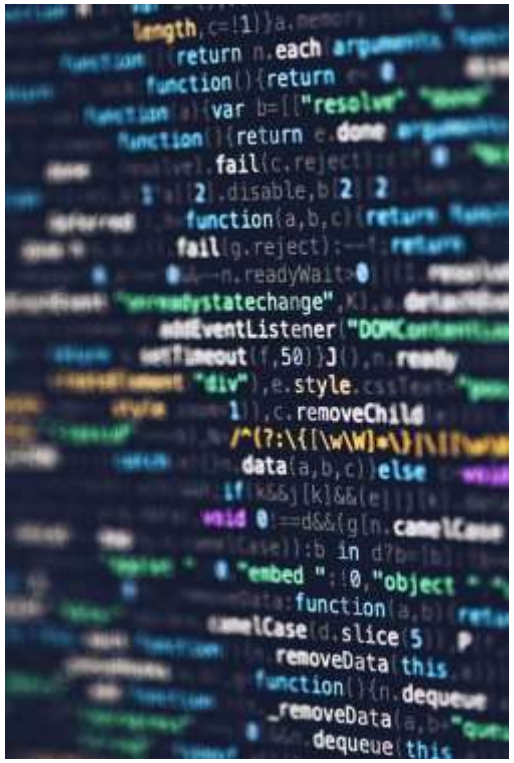
•Facilita la descripción de los procesos que sigue un ordenador cuando ejecuta un programa en ese lenguaje específico.
•Aporta una visibilidad que ayuda a comprender mejor lo que está haciendo un programa.
•Permite **conocer el significado de los lenguajes de programación** y cómo pueden surgir operaciones no deseadas



Semántica

Tipos de semántica en programación:

- Semántica operativa u operacional.
- Semántica denotacional
- Semántica axiomática

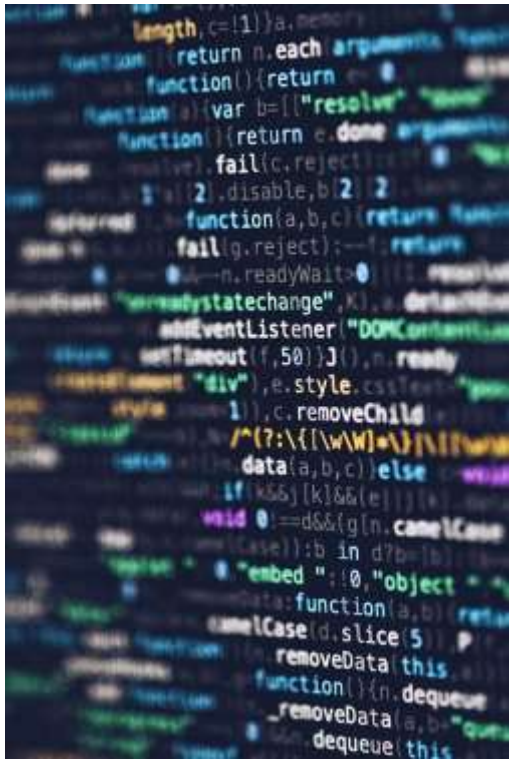


Semántica

Tipos de semántica en programación:

Semántica operativa:

La semántica operativa utiliza la idea de que **los lenguajes son máquinas abstractas** y la evaluación de un programa es una serie de transiciones de estado desde un estado inicial a un estado final.
Las funciones de transición definen cómo transitan los estados al siguiente, si lo hay. Si no existe el siguiente estado, la máquina completa su evaluación con éxito o se enfrenta a un error de tiempo de ejecución y se atasca, el programa se detiene.
Cada término en el programa tiene algún significado, y su forma finaliza cuando se completan las **transiciones de estado**. Las transiciones de estado pueden ser de uno o varios pasos, pequeños o grandes. Dado que la semántica operativa tiene un estilo basado en el comportamiento abstracto de la máquina, los pasos son útiles como referencia para la implementación.

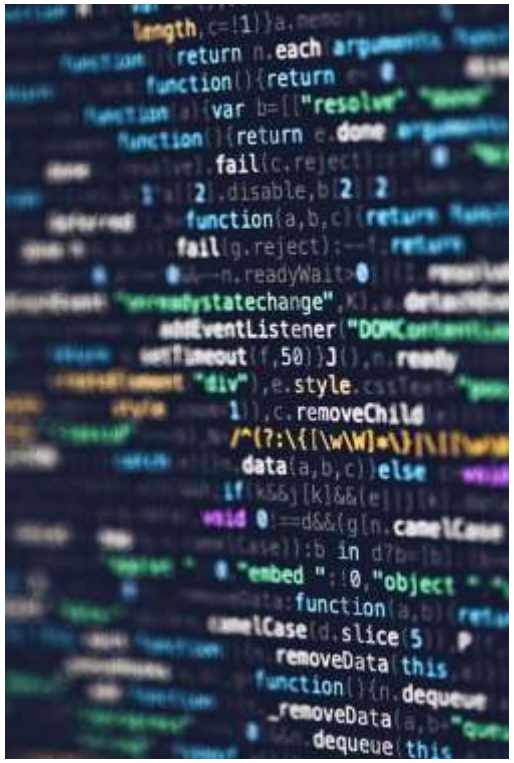


Semántica

Tipos de semántica en programación:

Semántica Denotacional:

La semántica denotacional utiliza la idea de que **los lenguajes son objetos matemáticos**. A diferencia de la semántica operativa, los detalles de evaluación e implementación se abstraen.
Una función de interpretación se define para asignar términos en un programa a elementos en dominios semánticos (también conocidos como su denotación), eliminando cualquier ocurrencia de la sintaxis original.
Los dominios semánticos están diseñados para **modelar características específicas del lenguaje** (teoría de dominio) y se pueden usar para mostrar instancias imposibles en un idioma

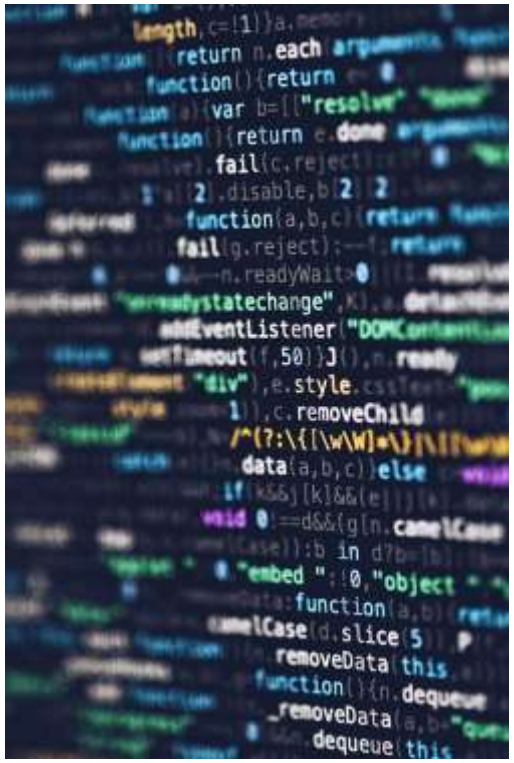


Semántica

Tipos de semántica en programación:

Semántica Axiomática:

En lugar de derivar leyes de definiciones de comportamiento operacional o denotacional, **las leyes mismas definen la semántica del lenguaje**. Esta inversión simplifica el razonamiento sobre un programa, lo que conduce a desarrollos en la verificación de software.
Se considera que dos implementaciones de programa diferentes con el mismo conjunto de afirmaciones iniciales y finales (leyes) tienen la misma semántica. Los términos que ocurren entre las afirmaciones solo se usan para probar las propias afirmaciones y no contribuyen a la semántica.
Las afirmaciones definen las relaciones entre las variables y otras partes móviles de un programa, y algunas de estas afirmaciones permanecen invariables durante la ejecución. Este es el importante concepto de invariancia que subyace a la semántica axiomática.

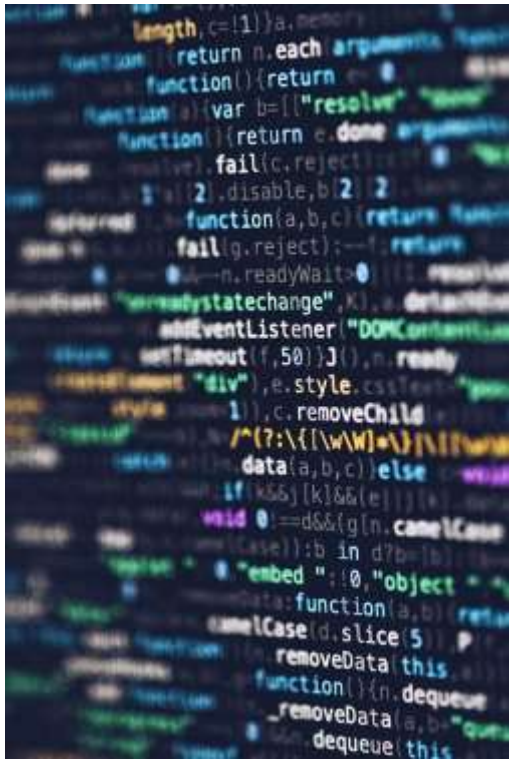


Algo de Semántica en Python:

Vamos a ver cómo Python interpreta la semántica en la práctica. En Python, podemos utilizar operadores como **+** para la adición. Veamos cómo funciona:

```
numero1 = 5
numero2 = 3
resultado = numero1 + numero2
print(resultado)
```

En este caso, Python entiende la semántica detrás de **+** como una operación de suma matemática y muestra el resultado correcto, que es 8, en la pantalla.



Ejemplos de Semántica en Python:

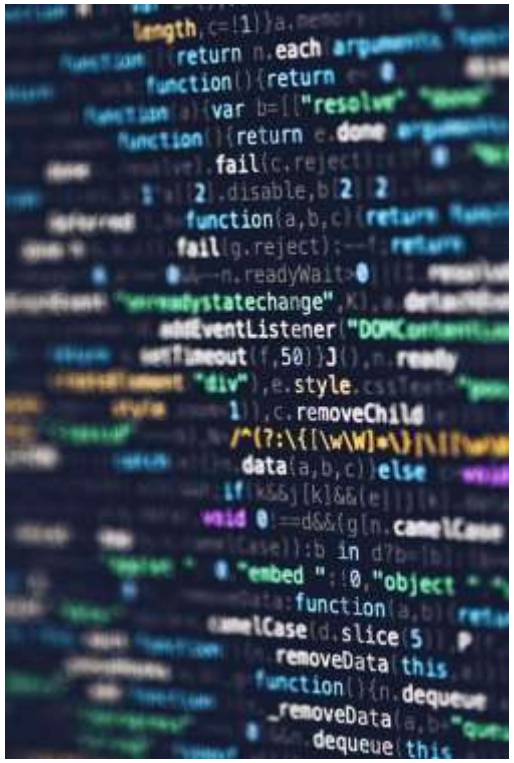
Consideremos otro ejemplo que involucra listas en Python. Supongamos que tenemos una lista de números y queremos agregar un número específico a esa lista. La sintaxis sería similar a esto:

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

Sin embargo, aquí cometemos un error de semántica. Python interpreta la adición de esta manera como una concatenación de listas en lugar de una adición de un número a la lista. La forma correcta de hacerlo sería:

```
numeros = [1, 2, 3, 4]
numero_a_agregar = 5
numeros.append(numero_a_agregar)
```

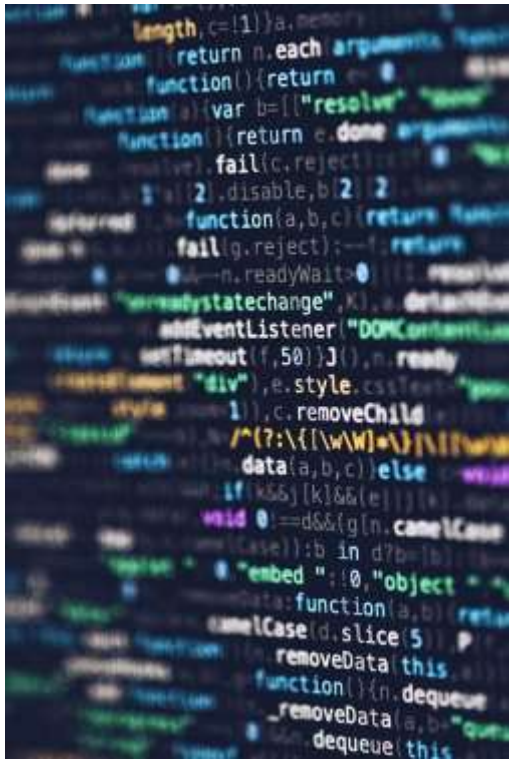
En este caso, utilizamos el método .append() para agregar el número a la lista, lo que tiene el significado adecuado en la semántica de Python. Si bien en ambos casos la sintaxis es correcta, la semántica solo lo es, para el objetivo que buscamos, en el segundo caso.



Semántica de programación en la práctica: paso a paso

Para entender de qué se trata la **semántica en programación**, lo mejor es ver cómo se plantea el proceso de prueba de los programas:

•Los programas se prueban mediante el **uso de casos de prueba**, lo que significa que alguien determina las posibles formas en que un usuario podría interactuar con el programa, tanto válidas como no válidas.
•Estos casos luego se ejecutan para ver cómo responde el programa.
•Cuando la semántica no coincida con el comportamiento esperado o deseado, es necesario cambiar algo antes del lanzamiento.
•Se necesitaría una **gran cantidad de tiempo y recursos** para probar todos los escenarios posibles, junto con los que tienen más probabilidades de causar problemas para el programa. De esa manera, se detectan tantos errores como sea posible lo más rápido posible.
•Mientras que un error de sintaxis podría impedir que un programa se ejecute o colapsarlo parcialmente, un **error de semántica** puede ser mucho menos dramático y algo tan simple como un botón que no realiza la acción que se supone que debe hacer, sino algo completamente distinto.
•Ambos tipos de errores deben **abordarse antes del lanzamiento**, por supuesto, pero el último suele ser más fácil de pasar por alto y más difícil de encontrar.



Errores de semántica

Los errores de semántica se refieren a la interpretación incorrecta de las sentencias.

Estos errores pueden ser difíciles de detectar y pueden generar resultados incorrectos o inesperados.

Los errores de semántica pueden incluir:

- **Tipos de datos incorrectos**: La asignación de un valor incorrecto a una variable puede generar un error de semántica.
- **Operadores incorrectos**: La utilización de operadores incorrectos puede generar un error de semántica.
  Por ejemplo, la utilización de un operador de asignación en lugar de un operador de comparación.
- **Funciones incorrectas**: La utilización de una función incorrecta puede generar un error de semántica.
  Por ejemplo, la utilización de una función de suma en lugar de una función de resta.

UNIVERSIDAD NACIONAL GUILLERMO BROWN

Ejemplos Prácticos….

El resaltado de sintaxis y el estilo de sangría se utilizan a menudo para ayudar a los programadores a reconocer elementos del código fuente. Este código de Python utiliza resaltado codificado por colores.

```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '    %s [label="%s' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= %s"];' % ast[1]
        else:
            print '"]'
    else:
        print '"];'
        children = []
        for n, child in enumerate(ast[1:]):
            children.append(dotwrite(child))
        print '    %s -> {' % nodename,
        for name in children:
            print '%s' % name,
```

Resaltado de sintaxis

El **resaltado de sintaxis**, a veces llamado **coloreado de sintaxis**, es una capacidad de algunos editores de texto para diferenciar elementos de texto (especialmente el llamado código fuente) mediante diversos colores o estilos tipográficos, dependiendo de las categorías sintácticas de sus términos, conforme a las reglas de algún lenguaje formal concreto.

Este resaltado se utiliza a modo de notación secundaria, habitualmente para mejorar la legibilidad del código fuente de programas o de textos escritos en algún lenguaje de marcado, permitiendo aumentar la productividad de los programadores

Ejemplo de resaltado de Sintaxis en C…

Presentación estándar	Resaltado de sintaxis

`/* Hello World */`
`#include <stdlib.h>`
`#include <stdio.h>`
 	
`int main()`
`{`
`    printf("Hello World\n");`
`    return 0;`
`}`
	`/* Hello World */`
	`#include <stdlib.h>`
	`#include <stdio.h>`
	
	`int main()`
	`{`
	`    printf("Hello World\n");`
	`    return 0;`
	`}`

UNIVERSIDAD NACIONAL GUILLERMO BROWN

Una Herramienta interesante .. Notepad++, (con archivo .php)

```
</script>
</head>
<body class="thebody" >
 <?php
include ("./lconnect.php");
$db = new lconnect;
$db->inicializar($dbhost,$user,$pwd,$databaseorigen,$dsn,$database);
$db->lconnect();
?>
<br>
<div class="container" >

        <!-- logo -->
        <?php $sql = "SELECT * FROM estructura where ide=7";
                $query = $db->lib->query($sql,$db->db);
                $row=$db->lib->fetch_array($query);
                echo trim($row['codigo']);?>
        <!-- fin logo -->
    <!--fin TOdo cabecera, cimuenza división del cuerpo ----------------------------
    <div class="span-24 last append-bottom">
        <!-- menu superior -->

      <?php $sql = "SELECT * FROM estructura where ide=4";
```

identifiquemos (C)

FUNCION - SINTAXIS

```
int suma (int a, int b)
{
   int r; r=a+b; return r;
}

int main ()

{
   int z; z = suma (5,3);
   cout << "El resultado es" << z;
   system("pause");
}
```

FUNCION - SINTAXIS

Ejemplo:

```
// Ejemplo de función
#include <iostream>
using namespace std;

int suma (int a, int b)
{
   int r; r=a+b; return r;
}

int main ()

{
   int z; z = suma (5,3);
   cout << "El resultado es" << z;
   system("pause");
```

Tipo de dato del valor retornado

Nombre con el que se llamará la función

Parámetros que actuan como variables Locales, pero externamente permiten Capturar los **argumentos** con los cuales Se realizara un computo o calculo.

Sentencias o instrucciones
Realizan la tarea de la función

Por ejemplo, en el lenguaje de programación Java, la siguiente sentencia tiene un error de sintaxis:

```
            if (x > 5
System.out.println("x es mayor que 5");
```

UNIVERSIDAD NACIONAL GUILLERMO BROWN

Encontrar el error de sintáxis

La sentencia correcta sería:

```
if (x > 5) {
System.out.println("x es mayor que 5");
}
```

Por ejemplo, en el lenguaje de programación Python, la siguiente sentencia tiene un error de…:

```
x = 5
if x > 10:
print("x es mayor que 10")
else:
print("x es menor o igual que 10")
```

```
x = 5
if x > 10:
print("x es mayor que 10")
else:
print("x es menor o igual que 10")
```

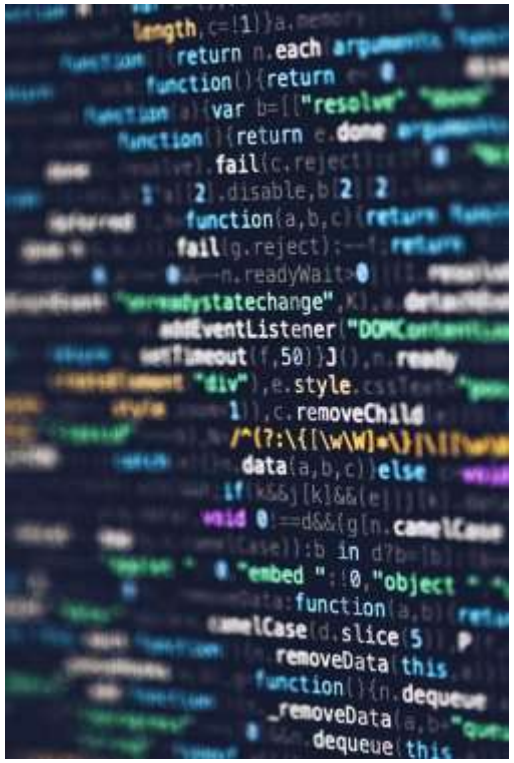
que tipo de error hay en la siguiente imagen:

```
int x = "5";
```

tiene un error de semántica:

```
int x = "5";
```

El error está en la asignación de un valor de cadena a una variable de tipo entero.



ver colab

- fin

Dudas?

Preguntas???????????