

Conceptos y Paradigmas de Lenguajes de Programación

Informe de Investigación

Grupo: 25

Lenguajes investigados:

→ Python

→ C

Autores:

Acuña Romina - 16415/6

Couchet Germán - 15776/5

Iriarte Fermín - 16368/6

Lucaroni Yanasú - 16377/7

Año 2020

1. Introducción

El siguiente informe corresponde a la asignatura Conceptos y Paradigmas de Lenguajes de Programación de la Facultad de Informática de Universidad Nacional de La Plata. El mismo tiene como objetivo definir y comparar los conceptos que atraviesan dos lenguajes de programación: C y Python. El documento se divide en secciones dedicadas a los conceptos generales, sintaxis, semántica, variables y parámetros, tipos admitidos y manejo de excepciones. Finalmente se encuentra un apéndice complementario y la bibliografía detallada que se consultó para su elaboración.

2. Sobre los lenguajes

Python: se caracteriza por ser un lenguaje de código abierto, interpretado, no compilado que utiliza tipado dinámico, fuertemente tipado. Es multiplataforma, lo cual permite hacer ejecutable su código fuente sobre varios sistemas operativos. Soporta varios paradigmas de programación como orientación a objetos, estructurada, programación imperativa y, en menor medida, programación funcional. Su formato de código (p. ej., la indentación) es estructural.

C: es un lenguaje de programación de nivel medio, que se beneficia de las ventajas de la programación de alto y bajo nivel. Se caracteriza por ser un lenguaje estructurado, que no depende del hardware, por lo que se puede migrar a otros sistemas. El desarrollo en C no tiene un enfoque específico, por lo que puede programarse desde un sistema operativo hasta una hoja de cálculo.

3. Conceptos que atraviesan a los lenguajes de programación aplicados a C y Python

3.1. Criterios de evaluación de lenguajes

Para evaluar lenguajes se deben establecer distintos criterios generales aplicables a cada uno de ellos. En este apartado se definen una serie de perspectivas a tener en cuenta a la hora de estudiar las características de un lenguaje.

a) Simplicidad y legibilidad:

Para evaluar la facilidad de escritura como lectura de un programa. En consecuencia, qué tan ameno resulta su aprendizaje y enseñanza.

Python: es un lenguaje de alto nivel, por lo que se acerca mucho al lenguaje natural. Gracias a ello, su sintaxis es muy simple y altamente legible, tanto que el código resultante se asemeja a pseudocódigos. Cuenta con tipado dinámico, gestor de memoria, gran cantidad de librerías disponibles y una gran potencia: todo esto hace que desarrollar programas en Python sea sencillo, rápido e incluso divertido. Por esos motivos se lo considera uno de los mejores lenguajes para aprender a programar.

C: el hecho de que se defina como un lenguaje de nivel medio, aumenta su complejidad. Su alcance amplio requiere de muchas componentes elementales, lo que también atenta contra su simplicidad. No se considera un lenguaje fácil de aprender ni enseñar debido a que requiere una continua práctica y un serio seguimiento si se quiere tener el control de los programas.

b) Claridad en los bindings:

Los elementos de los lenguajes de programación pueden ligarse a sus atributos o propiedades en diferentes momentos y debe quedar claro.

Python: funciona con tipado dinámico, es decir que las variables no se declaran explícitamente, sino que se crean cuando se realiza una asignación. Por lo tanto, su contenido puede ir cambiando durante la ejecución del programa. En este sentido, no es sencillo identificar el tipo de dato que almacena, ya que este se detecta según el tipo del valor que toma en un momento dado.

C: Al ser un lenguaje de tipado estático, las variables deben declararse explícitamente indicando el tipo de valor que almacenará. Esto hace que resulte más fácil saber el tipo de dato que contiene.

```
#include <stdio.h>

int main(){
    char saludo[]="hola";
    printf("%s\n",saludo);
    int a= 10;
    printf("%i",a);
    return 0;
}
```

Imagen 1.1: Variables en C.

```
print("esto es una suma")
numero_1=2
numero_2=4
resultado=numero_1+numero_2
print(resultado);
resultado="hola mundo"
print(resultado)
```

Imagen 1.2: Definiciones de variables en Python.

Estos fragmentos de código muestran cómo en C cada variable declara su tipo y haciendo posible reconocer a simple vista qué imprime cada *print*. En cambio, en Python, la misma variable que se utiliza para imprimir el resultado de una expresión, se utiliza para imprimir un *string*. Esto último resulta más confuso al momento de identificar que imprime cada *print*.

c) Confiabilidad:

Este criterio está relacionado con el chequeo de tipos y el manejo de excepciones que ofrece el lenguaje.

Python: el tipado dinámico puede traer desventajas. Por ejemplo, si se intenta comparar dos valores de tipos distintos, se producirá un error. Y como el chequeo de tipos en Python no se realiza hasta llegado el momento de la ejecución, esta falla no podría detectarse antes y se interrumpiría la ejecución del programa. Esto le quita confiabilidad al lenguaje y lo hace menos seguro.

Python ofrece mecanismo para el manejo de excepciones¹ mediante el bloque *try/exception*. El mismo se detallará más adelante en este informe.

C: el tipado estático hace que los errores de tipos se reduzcan, esto le brinda mayor seguridad al lenguaje. Por otro lado, C no ofrece mecanismo para el manejo de excepciones, aunque se puede construir algo parecido (pero limitado).

```
try:
    num = 2
    print (no_existe)
except NameError:
    print ("La variable no existe")
finally:
    print ("Programa finalizado")
```

En la imagen de ejemplo, se intenta imprimir el valor de una variable que no fue declarada. El intérprete lo detecta e imprime el texto definido en el bloque *except*. Por último imprime lo contenido en el bloque *finally*.

Imagen 1.3: Ejemplo de código manejando excepciones en Python.

¹ interrupción de ejecución por una falla en el programa.

d) Soporte:

Relacionado a la accesibilidad que se ofrece para quien esté interesado en usarlo o instalarlo.

Python: ofrece gran soporte puesto a que se caracteriza por ser de código abierto y multiplataforma. Tiene mucha documentación disponible que en su mayor parte fue aportada por la comunidad de usuarios. Además, para utilizarse solo se necesita un editor de texto y su intérprete.

C: también es multiplataforma. No es de código abierto, la comunidad no tiene libertades para modificarlo, lo cual lo hace menos fuerte en este punto. Por otro lado, es uno de los lenguajes más antiguos, por lo que se encuentra disponible gran cantidad de documentación como librerías, manuales y tutoriales.

e) Abstracción:

Relacionada a la capacidad de definir y usar estructuras u operaciones complicadas de manera que sea posible ignorar detalles de implementación.

Python: El paradigma de orientación a objetos tiene una muy buena capacidad de abstracción gracias al encapsulamiento. Este permite detallar las tareas que podrá realizar un objeto sin mostrar en detalle el trabajo de llevar a cabo cada acción.

C: al ser de programación procedural se debe prestar más atención a la implementación porque cada función declarada podría no ser tan clara como se desea. Esto genera un impacto negativo respecto a este criterio.

```
class Humano:
    def __init__(self, edad):
        self.edad=edad

    def hablar(self, mensaje, edad):
        print (mensaje, edad)

pedro= Humano(26)
raul = Humano(21)
pedro.hablar('Hola soy pedro y tengo', pedro.edad)
raul.hablar('Hola pedro, soy raul y tengo', raul.edad)
```

Imagen 1.4: Definición de una clase en Python.

En este ejemplo de Python se puede ver cómo utilizando programación orientada a objetos, el cuerpo del programa principal solo llama a los métodos definidos en la clase. De esta manera, el programa principal resulta mucho más simple.

En esta porción de código de C, se observa que el programa principal calcula el máximo valor de un vector utilizando la función calcularMaximo, o sea en el programa principal simplemente dispone de una función para realizar dicho cálculo, pero no se especifica cómo se hace, ese funcionamiento está definido en la función calcularMaximo.

```
#include<stdio.h>

int calcularMaximo(int vector[], int numeroDeElementos){

    int i, max;
    max=0;
    for (i=1; i<numeroDeElementos; i++){
        if (vector[i]>max)
            max=vector[i];
    }
    return max;
}

int main(){
    int vectorDeEnteros[100];
    int numeroDeElementos=100;
    max =calcularMaximo (vectorDeEnteros, numeroDeElementos);
    return 0;
}
```

Imagen 1.5: Funciones en C.

f) Ortogonalidad:

Este punto significa que un conjunto pequeño de constructores primitivos, puede ser combinado en número relativamente pequeño a la hora de construir estructuras de control y datos. Cada combinación es legal y con sentido

Python: “In Smalltalk and Python all values are objects and all types are reference types. Thus, these languages use reference semantics only, and collections of objects are treated in an orthogonal manner”-Programming Languages Principles and Practice Third Edition-Kenneth C. Louden.

```
#Creacion de una colección:
lista= ["A","B","C"]
tupla= ("A","B","C")
diccionario= {"1":A,"2":B,"3":C]

#Acceso a sus elementos
lista[0]
tupla[0]
diccionario["1"]
#Eliminación de elementos
del lista[0]
del diccionario["1"]
```

Esto indica que el comportamiento de las colecciones en Python se tratan de maneras similares, por lo tanto, una vez que se aprende a manejar una lista, también se aprende a manejar tuplas y diccionarios.

Imagen 1.6: Porción de código en Python.

```
struct Persona1{
    char nombre[20];
    int edad;
    struct Persona madre;
    struct Persona padre;
};
struct Persona2{
    char nombre[20];
    int edad;
    struct Persona* madre;
    struct Persona* padre;
};
```

C: se dice que es poco ortogonal debido a ciertos comportamientos. Uno de estos es por ejemplo las dos estructuras record y array. El record puede ser devuelto de una función, en cambio, el array no, siendo que ambas son estructuras de datos. Otro ejemplo es que los datos de las estructuras pueden ser de cualquier tipo excepto void y una estructura del mismo tipo. Estos comportamientos afectan negativamente a la Ortogonalidad del lenguaje.

En este ejemplo, la Persona1 no está bien declarada porque se contiene a si misma, una solución a esto es definir un puntero que apunte a una variable del mismo tipo como es en Persona2.

Imagen 1.7: Porción de código en C.

g) Eficiencia:

Un lenguaje eficiente debe administrar los recursos computacionales (memoria y procesador) de manera óptima y no sobreexigir esfuerzo humano.

Según una tabla realizada a partir las investigaciones de un equipo de portugueses a través del estudio [Energy Efficiency across Programming languages](#). Esta [tabla](#) mide energía consumida, tiempo de ejecución y memoria utilizada. C es considerado el lenguaje más eficiente respecto a los 3 campos. Python, en cambio, es de los que más energía y tiempo requieren pero tiene relativamente un buen uso de memoria.

3.2 La sintaxis

Esta sección está dedicada a definir el concepto y comparar las características sintácticas destacables de ambos lenguajes.

Se denomina sintaxis al un conjunto de reglas que define cómo combinar componentes para formar sentencias válidas dentro de un programa. Que sea “válida” quiere decir que el código de alto nivel puede traducirse a código de bajo nivel. De lo contrario, el compilador o intérprete encargado de esta tarea, no es capaz de entender la instrucción para realizar la conversión.

Los conjuntos de reglas se dividen en dos tipos. Por un lado existen las reglas “léxicas”, que definen la manera de formar las *word*² y por el otro, están las reglas sintácticas, para determinar la estructura de una sentencia.

También la sintaxis se clasifica en tres tipos.

- Abstracta: define a la estructura.
- Concreta: define las reglas léxicas.
- Pragmática: se refiere a la parte práctica.

Las posibles combinaciones para formar tanto *words* como sentencias, son infinitas. Por esta razón, para expresar la definición de cada regla, se utilizan notaciones libres de contexto para expresar la gramática. En este informe no se detalla de esa manera, pero sí se exponen ejemplos concretos de código por cada aspecto para realizar un contraste entre distintas características de ambos lenguajes.

- IDENTIFICADORES:

Ejemplo en Python:

```
estoEsUna_variable1 = 'variable'
ESTOesotr4v4riAbl_3 = 'otra variable'
```

Ejemplo en C:

```
una_Variable = 'hola'
unNUMEro123422 = 56
```

Si bien se exponen los ejemplos divididos, C como Python tienen la misma sintaxis para los identificadores. Ambos son *case-sensitive*, permiten todas las letras y dígitos y excluyen caracteres especiales, a excepción del “_”. La diferencia radica en que, si bien no limitan la longitud, en C solamente se tienen en cuenta los primeros 31 caracteres.

COMENTARIOS:

Ejemplo en Python:

```
# este es un comentario con una línea
```

```
""" este
```

² Conjunto de caracteres y palabras necesarias para construir expresiones, sentencias y programas.

es
otro con varias líneas *"""*

Ejemplo en C:

```
// Este es un comentario con una única línea  
/* este es  
un comentario con  
varias líneas*/
```

Se puede ver que son iguales en sintaxis abstracta pero distintas en sintaxis concreta. Pragmáticamente son similares, ya que si se quiere agregar una línea más a un comentario, hay que cambiar el comienzo o agregar otro carácter de línea única.

DELIMITACIÓN DE BLOQUES

Ejemplo en Python:

```
if (x == 10):  
i += i  
print('se suma uno a la variable i')
```

Ejemplo en C:

```
if (x == 10) {  
i += i;  
printf('se suma uno a la variable i'); }
```

En este sentido, son similares en sintaxis abstracta ya que el bloque tiene la misma estructura: expresión, bloque. Concretamente difieren: en C las sentencias pertenecientes a un bloque se encapsulan con llaves exceptuando el caso en que se tenga un bloque de una única sentencia, en cambio, en Python un bloque se delimita con la indentación. De esta manera, todas las líneas indentadas a la misma altura, pertenecen al mismo bloque. Pragmáticamente son similares, ambas podrían adoptar un comportamiento no esperado si se indenta incorrectamente o no se agrega la instrucción dentro de las llaves.

DELIMITACIÓN DE INSTRUCCIONES

Ejemplo en Python:

```
print('esta es una sentencia')
```

Ejemplo en C:

```
printf('esta es una sentencia');
```

C define como obligatorio el uso de punto y coma (";") para delimitar una instrucción de otra. Mientras que Python lo define como optativo ya que en realidad no produce cambios internamente, solo existe la opción si al programador le sirve vistosamente.

En este sentido, son distintos concretamente pero similares en la parte abstracta.

Pragmáticamente en Python esto en principio puede generar confusión ya que uno podría pensar que no se distingue con suficiente claridad dónde termina una instrucción. Lo cierto es que el salto de línea es lo que lo delimita. Con lo cual, el código resultante queda más ordenado dado que obligatoriamente cada instrucción se escribe en una línea y no se mezcla con otras. En cambio, en C uno podría escribir un programa complejo en una sola línea, lo cual sería poco legible. Por último, si el programador olvida cerrar la instrucción con “;”, se detectaría un error del mismo modo sucedería de olvidarse una indentación en Python.

DECLARACIÓN DE VARIABLES

Ejemplo en Python:

```
var1 = 1;
```

Ejemplo en C:

```
int var0, var1 = 1;
```

En este sentido difieren en sintaxis abstracta, ya que no tienen la misma estructura: C, al ser un lenguaje con tipado estático, determina que en la declaración de una variable, se especifique el tipo. Python por su parte, funciona con tipado dinámico, con lo cual la declaración de variables no necesita especificar el tipo. En sintaxis concreta son similares ya que utilizan el mismo operador para la asignación pero en C puede declararse más de una con el mismo valor (como se ve en el *Ejemplo 2*), en cambio en Python no es posible. Con respecto a la pragmática, podría cometerse el error de no haber declarado el tipo en C, lo cual en Python no hace falta.

El hecho de estos elementos que podrían o no estar, que son opcionales, le suman ambigüedad al lenguaje, lo que en realidad debería evitarse.

DECLARACIÓN DE FUNCIONES

Ejemplo en Python:

```
def suma (numero, otroNumero):  
numero += otroNumero  
return numero
```

Ejemplo en C:

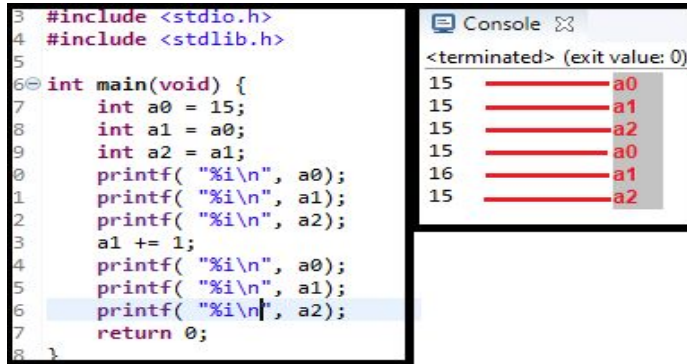
```
int suma (int numero, otroNumero) {  
numero = numero + otroNumero;  
return numero;}
```

El concepto del tipado también aplica en las funciones. En C, si una función tiene valor de retorno, en el encabezado de la declaración debe indicarse el tipo de ese valor de retorno. Lo mismo con los parámetros formales. Caso contrario, se indica con la palabra reservada “void”. En Python esto no es necesario. Por lo tanto, una declaración en C queda más cargada de componentes que en Python. Demasiados componentes juntos hacen que la legibilidad se pierda.

3.3 Semántica

La semántica describe el significado de los símbolos, palabras y frases de un lenguaje.

Python tiene reglas semánticas similares a las de C puesto a que fue desarrollado en C. En el siguiente ejemplo se puede ver que se comportan de la misma manera:



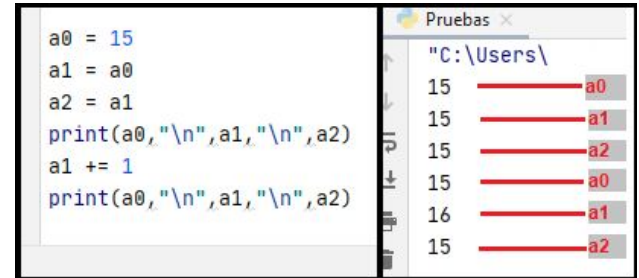
The image shows a C code editor window with the following code:

```
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main(void) {
7     int a0 = 15;
8     int a1 = a0;
9     int a2 = a1;
10    printf( "%i\n", a0);
11    printf( "%i\n", a1);
12    printf( "%i\n", a2);
13    a1 += 1;
14    printf( "%i\n", a0);
15    printf( "%i\n", a1);
16    printf( "%i\n", a2);
17    return 0;
18 }
```

Next to the code is a console window showing the output:

```
<terminated> (exit value: 0)
15
15
15
15
16
15
```

Imagen 2.1: Porción de código en C.



The image shows a Python code editor window with the following code:

```
a0 = 15
a1 = a0
a2 = a1
print(a0, "\n", a1, "\n", a2)
a1 += 1
print(a0, "\n", a1, "\n", a2)
```

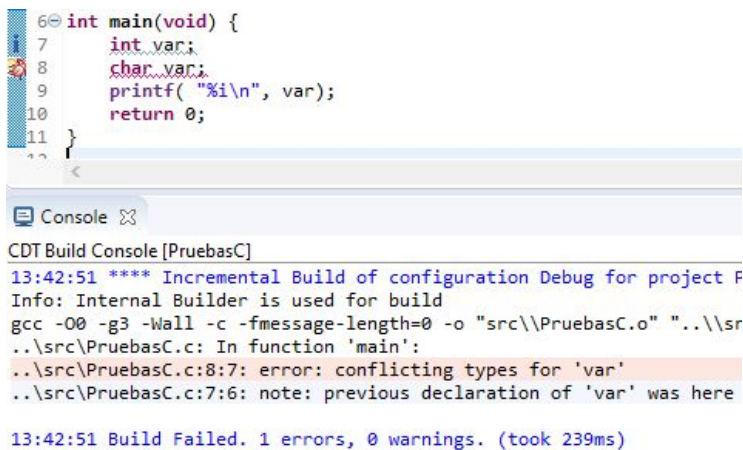
Next to the code is a console window showing the output:

```
"C:\Users\
15
15
15
15
16
15
```

Imagen 2.2: Porción de código en Python.

Se declaran tres variables. A cada una se le asigna el valor de la anterior declarada. Lo que ocurre es que a “a1” se le asigna una copia del valor de la referencia de “a0” y a “a2” el de “a1”. Por lo tanto, más adelante cuando “a1” aumenta en uno su valor, no modifica a “a0” ni a “a2”.

Al mismo tiempo, presentan diferencias por características que se detallaron anteriormente:



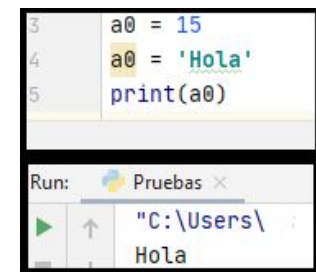
The image shows a C code editor window with the following code:

```
6 int main(void) {
7     int var;
8     char var;
9     printf( "%i\n", var);
10    return 0;
11 }
```

Below the code is a console window showing the output of the CDT Build Console:

```
CDT Build Console [PruebasC]
13:42:51 **** Incremental Build of configuration Debug for project F
Info: Internal Builder is used for build
gcc -O0 -g3 -Wall -c -fmessage-length=0 -o "src\PruebasC.o" "..\sr
..\src\PruebasC.c: In function 'main':
..\src\PruebasC.c:8:7: error: conflicting types for 'var'
..\src\PruebasC.c:7:6: note: previous declaration of 'var' was here
13:42:51 Build Failed. 1 errors, 0 warnings. (took 239ms)
```

Imagen 2.3: Porción de código en C.



The image shows a Python code editor window with the following code:

```
5 a0 = 15
4 a0 = 'Hola'
5 print(a0)
```

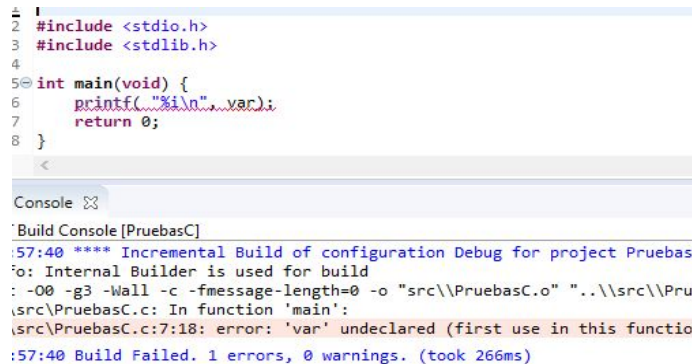
Below the code is a console window showing the output:

```
Run: Pruebas
"C:\Users\
Hola
```

Imagen 2.4: Porción de código en Python.

En C, no se puede declarar dos variables con el mismo nombre y no se puede ejecutar el programa. En cambio, en Python, no significa que se declare dos veces la misma variable, sino que se sobrescribe el valor anterior y se queda con el último asignado.

3.3.1 Semántica Estática: la semántica estática se trata de comprobar dichos aspectos en el momento de compilación.

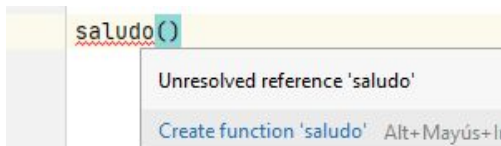


```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(void) {
6     printf("%i\n", var);
7     return 0;
8 }
```

Build Console [PruebasC]
:57:40 **** Incremental Build of configuration Debug for project PruebasC
Internal Builder is used for build
: -O0 -g3 -Wall -c -fmessage-length=0 -o "src\\PruebasC.o" "..\\src\\PruebasC.c"
src\\PruebasC.c: In function 'main':
src\\PruebasC.c:7:18: error: 'var' undeclared (first use in this function)
:57:40 Build Failed. 1 errors, 0 warnings. (took 266ms)

En este ejemplo en C, se intenta imprimir el valor de una variable que no se declaró. Por lo tanto, cuando se compila el código, se detecta un error, con lo cual no se puede ejecutar el programa.

Imagen 2.5: Porción de código en C.



```
saludo()
saludo()
```

Unresolved reference 'saludo'

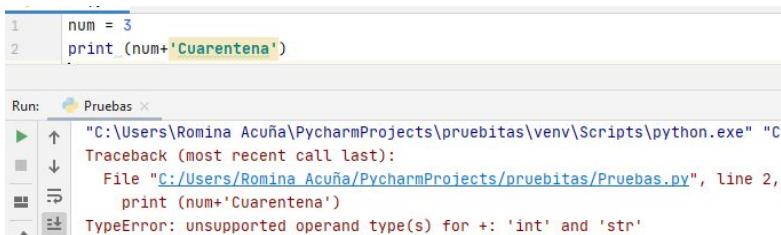
Create function 'saludo' Alt+Mayús+I

Como se mencionó anteriormente, Python cuenta con un intérprete en lugar de un compilador, por lo que claramente no hay “detección de errores en momento de compilación”. Pero sí aplica el concepto de semántica estática en un momento anterior a la ejecución. Como se puede apreciar en el siguiente ejemplo, donde se intenta llamar a una función que no fue declarada en ese momento, conflicto que se advierte antes de que se ejecute.

Imagen 2.6: Porción de código en Python.

3.2.2 Semántica dinámica: la semántica dinámica es la que se encarga de comprobar el cumplimiento de las reglas en el momento de ejecución.

En Python la mayoría de los errores se detectan en tiempo de ejecución, por lo que este concepto se ejemplifica mejor utilizando código en este lenguaje:



```
1 num = 3
2 print(num+'Cuarentena')
```

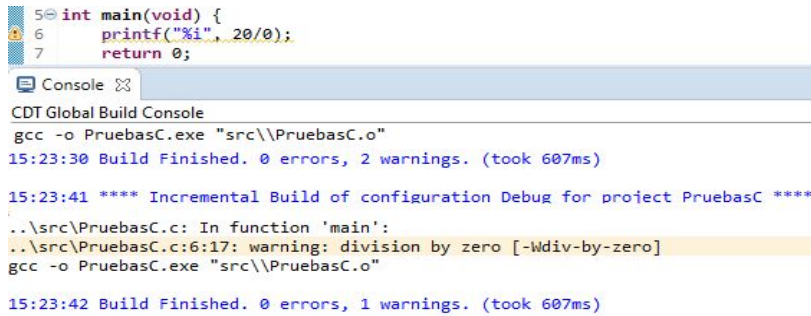
Run: Pruebas

Traceback (most recent call last):
File "C:/Users/Romina Acuña/PycharmProjects/pruebas/Pruebas.py", line 2, in
print(num+'Cuarentena')
TypeError: unsupported operand type(s) for +: 'int' and 'str'

Antes de ejecutarse, no se detectó error alguno. Cuando se corre el programa, finaliza la ejecución por un error semántico. Como se detalla en la consola, se combinan dos tipos de datos incompatibles en una expresión y no hay reglas que lo permitan o resuelvan.

Imagen 2.7: Porción de código en Python.

En C, también se aplica el concepto.



```
5 int main(void) {
6     printf("%i", 20/0);
7     return 0;
}

Console
CDT Global Build Console
gcc -o PruebasC.exe "src\\PruebasC.o"
15:23:30 Build Finished. 0 errors, 2 warnings. (took 607ms)

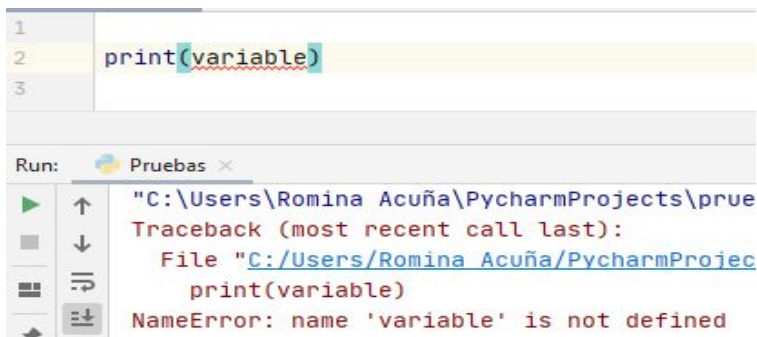
15:23:41 **** Incremental Build of configuration Debug for project PruebasC ****
..\src\PruebasC.c: In function 'main':
..\src\PruebasC.c:6:17: warning: division by zero [-Wdiv-by-zero]
gcc -o PruebasC.exe "src\\PruebasC.o"
15:23:42 Build Finished. 0 errors, 1 warnings. (took 607ms)
```

En este ejemplo, se intenta dividir por cero. Esta operación no está permitida, pero el error no se detectó en momento de compilación y el programa pudo ejecutarse. De todas maneras, no se imprimió nada en la consola pero sí se puede apreciar el error.

Imagen 2.8: Porción de código en C.

C y Python aplican los conceptos de semántica estática y dinámica en la detección de errores, pero lo hacen de manera distinta. Como se indicó anteriormente, las variables no declaradas en C se detectan con la semántica estática. No pasa lo mismo con Python:

Como en la imagen 2.5 del ejemplo en C, se intenta imprimir una variable no declarada. Antes de ejecutarse, no se detectó error. Al correr el programa, terminó con el error que se muestra en consola.



```
1
2 print(variable)
3

Run: Pruebas x
"C:\Users\Romina Acuña\PycharmProjects\prue
Traceback (most recent call last):
  File "C:/Users/Romina Acuña/PycharmProjec
    print(variable)
NameError: name 'variable' is not defined
```

En el caso de Python, se detecta con la semántica dinámica. De esta manera ocurre con la mayoría de los errores, debido a la diferencia en el modo de traducir el código: el intérprete y el compilador.

Imagen 2.9: Porción de código en Python.

4. Variables

Las variables en un programa son abstracciones identificadas con un nombre único que representan una celda de memoria o una colección de ellas. Conceptualmente, una variable se caracteriza como una tupla que contiene los siguientes atributos:

- Nombre: string de caracteres que se usa para referenciar a la variable.
- Alcance: rango de instrucciones en el que se conoce el nombre.
- Tipo: valores y operaciones.
- L-value: lugar de memoria asociado con la variable (tiempo de vida).
- R-value: valor codificado almacenado en la ubicación de la variable.

A continuación, se presenta un ejemplo en C para ver en detalle cada uno de estos atributos y sus posibles valores.

Para comprender el cuadro:

En C, el “L-valor” puede ser “automático” (variables definidas en un bloque), “estático” (variables con especificación “static” en su declaración) o “dinámico” (punteros).

Por otro lado, las variables declaradas globales y estáticas se inicializan por defecto (sin necesidad de una inicialización explícita por el programador) en cero. Las variables automáticas, se inicializan por defecto con un valor indefinido, también llamado “basura”.

Las variables que se definen por fuera de todas las funciones se conocen durante todo el programa (globales). Sin embargo, cuando se declara una variable con el mismo nombre en algún bloque de código, dejan de ser “visibles” como globales y se ve limitado su alcance, dado que fue definida una variable con su mismo nombre.

```
1  int var0;
2  static int var1;
3  extern int var2;
4
5  int aumentar() {
6      var1 += 1;
7      return 0;
8  }
9
10 int decrementar() {
11     register int var3;
12     int var1;
13     var1 -= var3;
14     return 0;
15 }
16
17 int main(void) {
18     int *ptr;
19     int var2;
20     var2 += 2;
21 }
```

Imagen 3.1: código en C.

Línea	Identificador	L-valor	R-valor	Alcance	Tiempo de vida
1	var0	Automática	0	2-21	1-21
2	var1	Estática	0	<3-11 16-21->	<1-21>
3	var2	Automática	0	3-18 21	1-21
11	var3	Automática	Indefinido	11-15	10-15
12	var1	Automática	Indefinido	12-15	10-15
19	var2	Automática	Indefinido	19-21	17-21
18	ptr	Dinámica	Indefinido	18-21	17-21
	aumentar			6-20	5-8
	decrementar			9-20	10-15
	main			18-21	17-21

5. Parámetros

Un programa puede estar dividido en *unidades*. Las mismas representan un fragmento que agrupa instrucciones para realizar una tarea específica. Estas unidades pueden comunicarse entre sí, cuya forma de hacerlo es a través de *parámetros*, o sea, datos que se envían entre ellas.

Al dato, ya sea valor u otra entidad que se pasa a una unidad, se lo llama “argumento” o “parámetro real”. La expresión (u otra constitución) que produce un argumento, es conocido como “parámetro” o “parámetro formal”. Es un identificador a través del cual un procedimiento puede acceder a un argumento.

A continuación se detallan las características de implementación del pasaje de parámetros de los dos lenguajes tratados en el documento.

Existen distintas convenciones para la implementación de pasaje de parámetros. Tanto Python como C, implementan “pasaje por valor”. En este tipo, la unidad invocada evalúa el argumento que se le está enviando, y el valor del mismo es utilizado para inicializar el parámetro formal correspondiente, que actúa como variable local en la función o procedimiento. En Python, particularmente, no existe la posibilidad de trabajar con pasaje por referencia (o sea, enviar la dirección real de memoria de una variable), en cambio, en C, podría simularse mediante el envío de punteros.

```
void cambiar (int x, int y){
    int aux;

    aux=x;
    x = y;
    y = aux;
    printf("En la funcion : x: %d y: %d \n",x,y);
}

int main (){
    int x,y;
void cambiar (int *x, int *y){
    int aux;

    aux=*x;
    *x = *y;
    *y = aux;
    printf("En la funcion : x: %d y: %d \n",*x,*y);
}

int main (){
    int x,y;
    x = 10;
    y = 5;
    printf("x: %d y: %d \n",x,y);
    cambiar(&x,&y);
    printf("x: %d y: %d \n",x,y);
    return 0;
}
```

En este ejemplo de C, la función cambiar intercambia los valores de X e Y, el primer printf, el segundo nos da como resultado que x = 5 y que y=10 pero el tercero incluso después de haber invocado a la función cambiar, nos da el mismo resultado que el primero, esto es debido a que los parámetros son pasados por valor.

```
x: 10 y: 5
En la funcion : x: 5 y: 10
x: 10 y: 5
```

```
x: 10 y: 5
En la funcion : x: 5 y: 10
x: 5 y: 10
```

Imagen 4.1: Código en C.

En este segundo ejemplo, los parámetros se envían “por referencia”, por lo tanto, una vez se invoca la función los valores se mantienen ya que lo que fue enviado a la función no fue una copia de los valores, sino su dirección de memoria.

Python: envía objetos que pueden ser “inmutables” o “mutables”. Si es inmutable actuará como por valor y, si es mutable, no se hace una copia sino que se trabaja sobre él. El pasaje de parámetros puede ser posicional o por nombre. Cuando es posicional, en la invocación a una unidad con varios argumentos, sus valores se corresponden con los parámetros según el orden en que se indicaron. Por otro lado, cuando es por nombre, se puede cambiar el orden pero en el llamado a la función, los argumentos deben indicar el mismo nombre que los parámetros formales.

```

1  def cambiar(x, y):
2      x, y = y, x
3      print('Dentro de la funcion "cambiar" A =', x, 'y B =', y)
4      return x, y
5
6
7  a = 10
8  b = 5
9  print('A =', a, 'y B =', b)
10 cambiar(a, b)
11 print('A =', a, 'y B =', b)
12 a, b = cambiar(a, b)
13 print('A =', a, 'y B =', b)
14

```

Imagen 4.2: Código en Python, uso de parámetros.

```

A = 10 y B = 5
Dentro de la funcion "cambiar" A = 5 y B = 10
A = 10 y B = 5
Dentro de la funcion "cambiar" A = 5 y B = 10
A = 5 y B = 10

Process finished with exit code 0

```

Imagen 4.3: Resultado de la ejecución del código de la imagen 4.2.

En C, todos los pasajes de parámetros se realizan por valor. Si se necesita por referencia, se utilizan punteros. C permite además el pasaje por *valor constante*, agregándole “const”. Además permite pasaje de funciones como parámetros.

6. Sistema de tipos de datos

Se define como “sistema de tipos” al conjunto de reglas que define el lenguaje para estructurar y organizar su colección de tipos de datos.

Un lenguaje se define como “fuertemente tipado” cuando puede prevenir errores de tipos ya sea en tiempo de ejecución como de compilación. De lo contrario, se lo considera un lenguaje débilmente tipado, como es el caso de C.

En el siguiente ejemplo

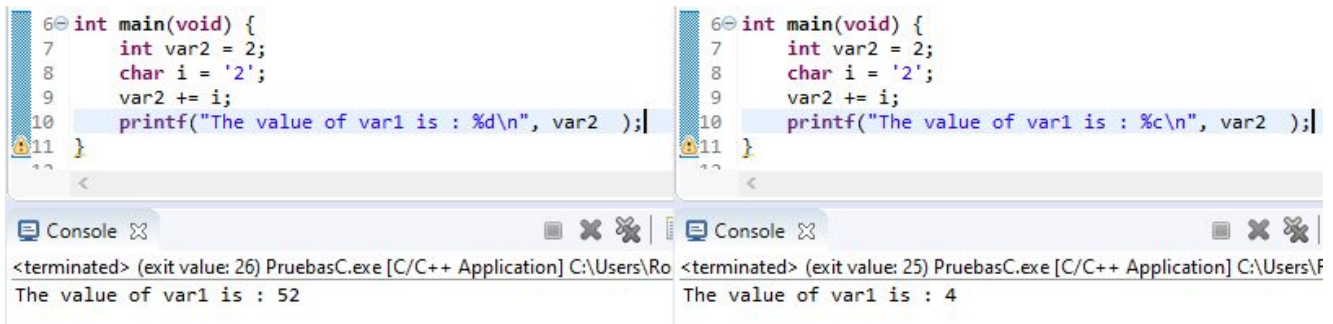


Imagen 5.1: Código en C imprime un decimal.

Imagen 5.2: Código en C imprime un caracter.

se puede apreciar cómo se permite una suma entre una variable de tipo caracter y una de tipo integer. En la imagen de la izquierda el resultado se imprime como un decimal (%d), devolviendo la suma de la representación ascii del caracter “2” (50) más el valor 2. En la imagen 4.2, como se imprime un caracter (%c), se devuelve lo que representa el 52 en la tabla ascii, o sea, el número 4 . Un lenguaje fuertemente tipado define restricciones sobre cómo las operaciones que involucran valores de diferentes tipos pueden operarse, como es el caso de Python. En el apartado dedicado a la semántica, la imagen 2.7 muestra la excepción que se produce al realizar la operación de suma entre una variable de tipo entero y una cadena de caracteres.

Por otro lado, C garantiza que a toda entidad definida para un tipo, se le asigne un valor que se adecúe al mismo. Pero esto no le otorga “mayor seguridad”, ya que en realidad lo que sucede es que a cualquier valor que se le asigne, se lo convierte en un dato de ese tipo. Por ejemplo, si a una variable declarada como “int” se le asigna una cadena de strings, la variable toma el valor ascii decimal de la representación de los caracteres que lo componen. A diferencia de Python, que, como se mencionó en secciones anteriores, permite que a una entidad se le pueda asignar un valor de cualquier tipo, pero esto no lo hace menos seguro. Esta distinción radica en una de las características principales de cada uno: la implementación de tipado estático y dinámico, respectivamente.

7. Manejo de excepciones

Se define como excepción a una condición inesperada o inusual, que surge durante la ejecución del programa y no puede ser manejada en el contexto local. Para estos eventos, un lenguaje de programación debería ofrecer herramientas para tratarlos.

Como lo hace Python, mediante el bloque “try-except”. Cuando surgen errores durante la ejecución del programa, se crean objetos especiales llamados “Exceptions”. El bloque try-except solicita a Python ejecutar un fragmento de código y, de crearse un objeto “exception”, el programador puede indicar cómo continuar para que el programa no finalice en la línea de conflicto. Este funciona como mecanismo “por terminación”, es decir que cada vez que se produce una excepción, se termina el bloque donde se

levantó y se ejecuta el manejador asociado, sin continuar con la ejecución del resto de las líneas siguientes dentro de ese bloque.

En el siguiente ejemplo se puede ver esto en detalle.

```
1 division = 2/0
2
3 print('Se realizó una división')
4
5 print('Acá continúa el programa.')
6
```

Run: Pruebas ×

"C:\Users\Romina Acuña\PycharmProjects\pruebas\pruebas.py"

Traceback (most recent call last):

File "C:\Users\Romina Acuña\PycharmProjects\pruebas\pruebas.py", line 1, in <module>

division = 2/0

ZeroDivisionError: division by zero

Process finished with exit code 1

Imagen 6.1: Python: división inválida sin manejo de excepciones.

```
1 try:
2     division = 2/0
3     print('Acá continúa el bloque try.')
4 except ZeroDivisionError:
5     print('No se puede dividir por cero.')
6 else:
7     print('Se realizó la división con éxito.')
8 finally:
9     print('Acá termina el manejador.')
10    print('Acá continúa el programa.')
```

Run: Pruebas ×

"C:\Users\Romina Acuña\PycharmProjects\pruebas\pruebas.py"

No se puede dividir por cero.

Acá termina el manejador.

Acá continúa el programa.

Process finished with exit code 0

Imagen 6.2: Python: división inválida con manejo de excepciones.

En la imagen 6.1, se realiza una división con denominador 0, lo cual levanta una excepción y produce la interrupción de la ejecución del programa. En la imagen 6.2, se realiza la misma operación con la implementación de try-except. En este, en lugar de finalizar en la línea del error, se ejecuta el bloque definido dentro del “except” y continúa con el resto de las líneas. También se expone lo mencionado con respecto al mecanismo “por terminación”, ya que no se ejecutó la línea 3 y no se imprimió en consola el string “Acá continúa el bloque try”. Por otro lado, se aprecia que a través de la línea “else:” puede definirse otro comportamiento dependiendo del error producido. Y por último, mediante el bloque definido luego de la palabra reservada “finally”, se puede definir un comportamiento que se ejecuta ante cualquier situación.

Como también se hizo mención anteriormente, en C no hay ningún mecanismo de excepciones. Se puede construir algo para su simulación utilizando las instrucciones “setjump” y “longjump”. Para implementar el modelo de terminación, hay que guardar y restaurar el estado de entrada al dominio de una excepción. De todas formas, utilizar setjump y longjump puede ser una mala idea, ya que al realizar el salto, todas las asignaciones, lockings, estructuras de datos inicializadas y demás, aún seguirán estando asignadas/bloqueadas o inicializadas después de realizar el salto, por lo que se debe ser muy cuidadoso al usar esta metodología para el manejo de excepciones.

Sabiendo esto, llegamos a la conclusión de que una manera óptima de manejar las excepciones sería la Terminación, ya que no hay que tener en cuenta el entorno en donde se haya encontrado la excepción, simplemente se termina el bloque en donde se ha generado, con su debida solución. En cambio en la reasunción, se debe analizar donde continuará el programa podría seguir en la siguiente línea o 5 líneas más abajo, incluso en otro bloque y en caso de que la excepción sea generada por un valor erroneo tambien debera cambiarse dicho valor, por eso se debe tener en cuenta el entorno en donde se genera la excepción.

8) Entrevista a un profesional:

Para el lenguaje C se contactó a la profesora de la cátedra de Seminario de Lenguajes orientado a C correspondiente a la Facultad de Informática de la Universidad Nacional de La Plata, Paola Amadeo.

La docente resaltó la eficiencia de C frente a otros lenguajes y la posibilidad de trabajar con la memoria.

Lo definió como un lenguaje hecho por y para programadores, lo que lo hace ideal para programadores expertos y cuenta con una curva de aprendizaje significativa.

Se le consultó sobre cómo había aprendido a utilizar este lenguaje, dijo que había recurrido a todo tipo de documentación como libros, videos, clases. Recalcó que es un lenguaje con muchísima documentación disponible para facilitar su aprendizaje.

Se preguntó también sobre qué parte del lenguaje le resultó más difícil de aprender, para ella la gestión de la memoria era la mayor complicación que presentaba este lenguaje.

Por último, se le pidió opinión acerca de las restricciones que encontró en C. Definió a C como un lenguaje con el que se puede hacer de todo, aunque cuando se trata de un desarrollo de software más tradicional para un cliente final, no es el más recomendable.

Conclusión final:

Luego de realizar este trabajo los autores aprendieron por qué es importante analizar cada aspecto de un lenguaje a la hora de elegir el óptimo para realizar una tarea específica.

Por ejemplo a la hora de aprender a programar desde un nivel básico es mejor utilizar un lenguaje que tenga una sintaxis y reglas semánticas simples, características que ayudan a la simplicidad y legibilidad de un lenguaje. También tener en cuenta si es un lenguaje compilado o interpretado, ya que los segundos pueden resultar más didácticos debido a que muestran los errores línea a línea, sin tener que llegar a compilar el programa. Por otro lado si lo que se quiere desarrollar un sistema más complejo y

rápido se debería utilizar un lenguaje compilado y con características que permitan una optima utilización de memoria y recursos de la computadora.

En particular sobre los dos lenguajes analizados los alumnos llegaron a la conclusión de que C es un lenguaje de programación más complejo por ser menos ortogonal y de nivel medio pero tiene una ventaja en cuanto a eficiencia tanto en tiempo de respuesta como en el uso de recursos (esto se debe, en parte, a su tipado estático). Respecto a Python, es un lenguaje dedicado a sistemas simples en los que no se centren tanto en la eficiencia ya que es interpretado y de tipado dinámico. Ambos lenguajes cuentan con la posibilidad de realizar distintos tipos de sistemas (desarrollo web, de aplicaciones, etc).

Por experiencia, los investigadores sabían que Python cuenta con mucho soporte en internet, posiblemente debido a su gran versatilidad y por ser de software libre, sin embargo desconocían que C también contaba con esas dos primeras características ya que es uno de los lenguajes más usados y de mayor antigüedad.

Bibliografía:

Burns A. y Wellings A., (2001) *Real-Time Systems and Programming Languages*, 3ª edición.

Couto, M., Cunha, J., Fernández, J., Pereira, R., Rui Rua, F., Saraiva, J. (2017) *Energy Efficiency across Programming Languages*, Vancouver, BC, Canada.

Freites González, R. (s/f) *Programar en PYTHON*. Recuperado de: <https://www.monografias.com/trabajos-pdf5/programar-python/programar-python.shtml>

Garrido, A. (S/F) *Abstracción en Programación*. España, Granada: Departamento de Ciencias de la Computación e I.A. ETS Ingeniería Informática Universidad de Granada.

Ghezzi, C. (1996), *Programming language concepts - Third edition*, USA, John Wiley and Sons.

González Duque, R. *Python para todos*. España: Creative Commons.

Kernighan, B., Ritchie, D., (1978) *The C Programming Language*, Second Edition. Estados Unidos, Prentice Hall.

Sugianto, S. (2013) *Concepts of Programming Languages*. Recuperado de: <https://stevanussugianto.wordpress.com/2013/03/04/20/>

Sebesta R., (2010) *Concepts of Programming Languages Tenth Edition*, Estados Unidos, Pearson.