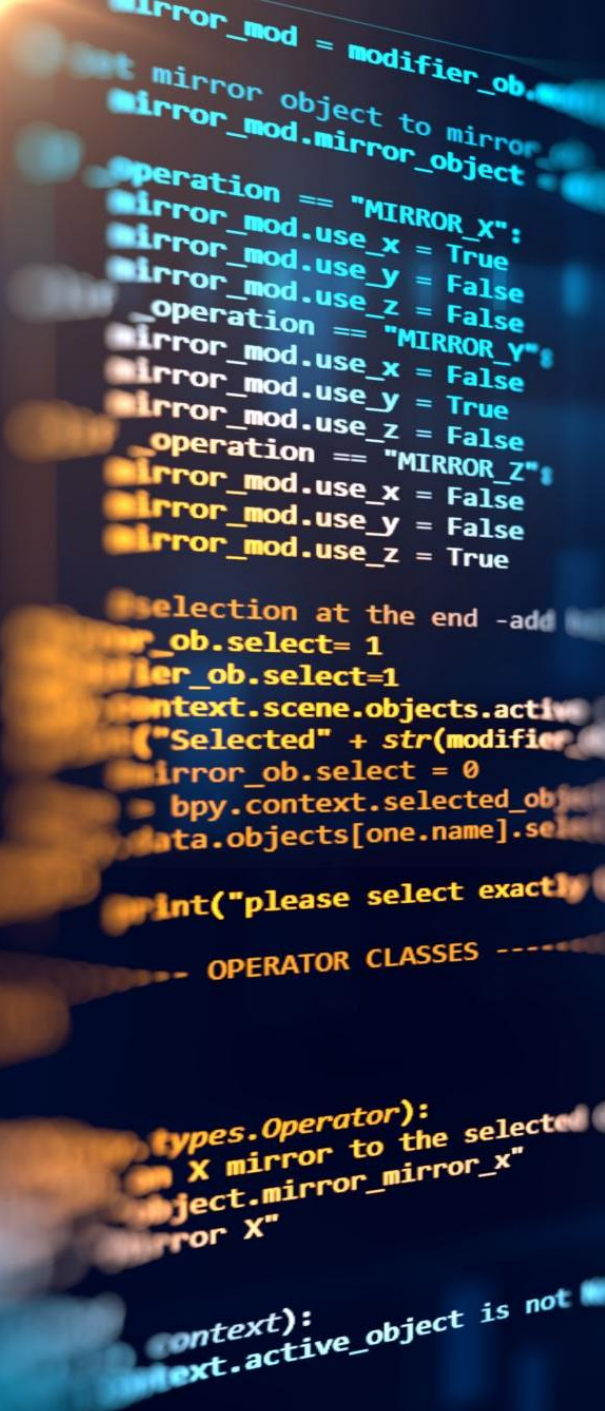


A detailed image of a snake, likely a corn snake, coiled against a dark background. The snake has a pattern of orange and yellow scales with darker bands. Its head is raised, showing a prominent red eye and a flicking red tongue. The lighting highlights the texture of the scales.

Fundamentos de Python

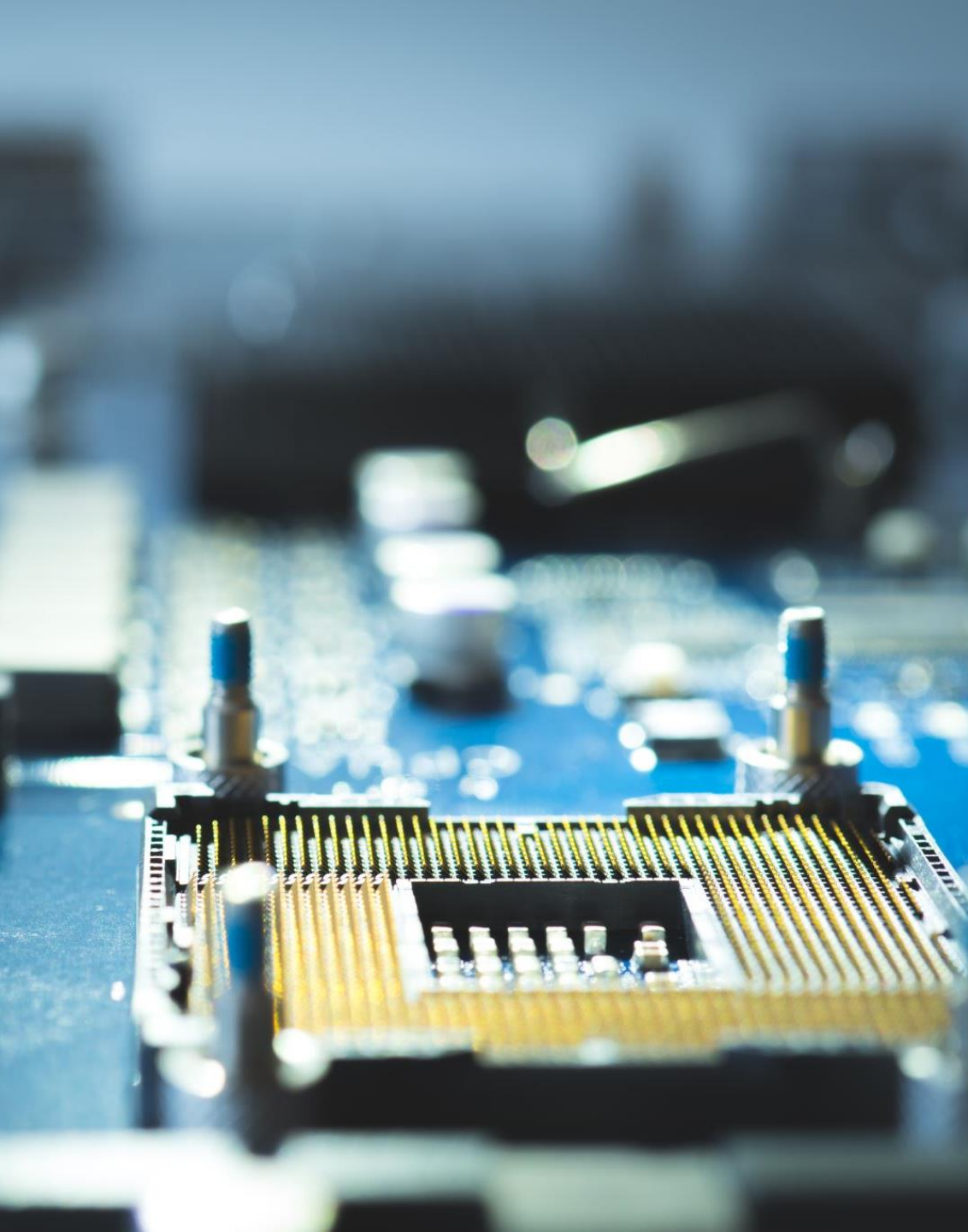
Plan de Estudio



```
mirror_mod = modifier_ob.  
set mirror object to mirror.  
mirror_mod.mirror_object =  
operation == "MIRROR_X":  
mirror_mod.use_x = True  
mirror_mod.use_y = False  
mirror_mod.use_z = False  
operation == "MIRROR_Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR_Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True  
selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob.  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
print("please select exactly  
-- OPERATOR CLASSES ----  
types.Operator):  
X mirror to the selected  
object.mirror_mirror_x"  
mirror X"  
context):  
context.active_object is not
```

Plan de Estudio

- Los conceptos universales de la programación informática;
- La sintaxis y la semántica del lenguaje Python;
- Habilidades prácticas para resolver desafíos típicos de implementación;
- Cómo utilizar los elementos más importantes de la biblioteca estándar de Python;
- Cómo instalar tu entorno de ejecución;
- Cómo diseñar, desarrollar, probar y depurar programas simples de Python.



Módulos

- • Módulo 1: Introducción a Python y programación informática;
- • Módulo 2: Tipos de datos, variables, operaciones básicas de entrada-salida y operadores básicos;
- • Módulo 3: Valores booleanos, ejecución condicional, bucles, listas y procesamiento de listas, operaciones lógicas y bit a bit;
- • Módulo 4: Funciones, tuplas, diccionarios, excepciones y procesamiento de datos.

Módulo 1: Sección 1 - Introducción a la programación



- ¿Cómo funciona un programa de computadora?



- Las computadoras pueden realizar tareas muy complejas, pero esta habilidad no es innata.



- Lenguajes naturales vs lenguajes de programación



- ¿Qué compone a un lenguaje?

¿Cómo funciona un programa de computadora?



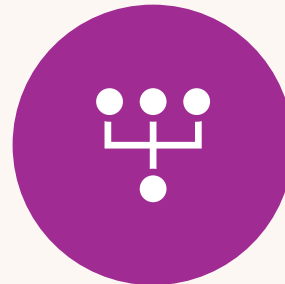
- Un programa hace que una computadora sea usable.



- Sin un programa, una computadora no es más que un objeto.



- Las computadoras solo pueden ejecutar operaciones extremadamente simples.



- Pueden realizar estas operaciones muy rápido y repetidamente.

Lenguajes naturales vs lenguajes de programación



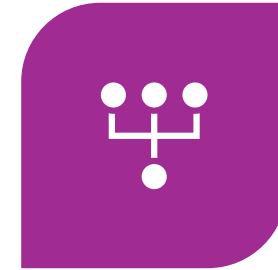
- UN LENGUAJE ES UN MEDIO PARA EXPRESAR PENSAMIENTOS.



- LAS COMPUTADORAS TIENEN SU PROPIO LENGUAJE: EL LENGUAJE MÁQUINA.



- LA COMPUTADORA RESPONDE SOLO A UN CONJUNTO DE COMANDOS PREDETERMINADOS.



- LOS LENGUAJES DE PROGRAMACIÓN PERMITEN A LOS HUMANOS EXPRESAR COMANDOS COMPLEJOS.

¿Qué compone a un lenguaje?



- Un Alfabeto: conjunto de símbolos utilizados para formar palabras.



- Un léxico: conjunto de palabras que ofrece el lenguaje.



- Una Sintaxis: conjunto de reglas para formar oraciones válidas.



- Una Semántica: reglas que determinan si una frase tiene sentido.

mónicos. Al principio, las instrucciones en un lenguaje ensamblador eran sólo representaciones mnemónicas de las instrucciones de máquina. Más adelante, se agregaron macro instrucciones a los lenguajes ensambladores, para que un programador pudiera definir abreviaciones parametrizadas para las secuencias de uso frecuente de las instrucciones de máquina.

Un paso importante hacia los lenguajes de alto nivel se hizo en la segunda mitad de la década de 1950, con el desarrollo de **Fortran** para la computación científica, **Cobol** para el procesamiento de datos de negocios, y **Lisp** para la computación simbólica. La filosofía de estos lenguajes era crear notaciones de alto nivel con las que los programadores pudieran escribir con más facilidad los cálculos numéricos, las aplicaciones de negocios y los programas simbólicos. Estos lenguajes tuvieron tanto éxito que siguen en uso hoy en día.

En las siguientes décadas se crearon muchos lenguajes más con características innovadoras para facilitar que la programación fuera más natural y más robusta. Más adelante, en este capítulo, hablaremos sobre ciertas características clave que son comunes para muchos lenguajes de programación modernos.

En la actualidad existen miles de lenguajes de programación. Pueden clasificarse en una variedad de formas. Una de ellas es por **generación**. Los *lenguajes de primera generación* son los lenguajes de máquina, los de *segunda generación* son los lenguajes ensambladores, y los de *tercera generación* son los lenguajes de alto nivel, como Fortran, Cobol, Lisp, C, C++, C# y Java. Los *lenguajes de cuarta generación* son diseñados para aplicaciones específicas como NOMAD para la generación de reportes, SQL para las consultas en bases de datos, y Postscript para el formato de texto. El término *lenguaje de quinta generación* se aplica a los lenguajes basados en lógica y restricciones, como Prolog y OPS5.

Otra de las clasificaciones de los lenguajes utiliza el término *imperativo* para los lenguajes en los que un programa especifica *cómo* se va a realizar un cálculo, y *declarativo* para los lenguajes en los que un programa especifica *qué* cálculo se va a realizar. Los lenguajes como C, C++, C# y Java son lenguajes imperativos. En los lenguajes imperativos hay una noción de estado del programa, junto con instrucciones que modifican ese estado. Los lenguajes funcionales como ML y Haskell, y los lenguajes de lógica de restricción como Prolog, se consideran a menudo como lenguajes declarativos.

El término *lenguaje von Neumann* se aplica a los lenguajes de programación cuyo modelo se basa en la arquitectura de computadoras descrita por von Neumann. Muchos de los lenguajes de la actualidad, como Fortran y C, son lenguajes von Neumann.

Un *lenguaje orientado a objetos* es uno que soporta la programación orientada a objetos, un estilo de programación en el que un programa consiste en una colección de objetos que interactúan entre sí. Simula 67 y Smalltalk son de los primeros lenguajes orientados a objetos importantes. Los lenguajes como C++, C#, Java y Ruby son los lenguajes orientados a objetos más recientes.

Los *lenguajes de secuencias de comandos (scripting)* son lenguajes interpretados con operadores de alto nivel diseñados para “unir” cálculos. Estos cálculos se conocían en un principio como “*secuencias de comandos (scripts)*”. Awk, JavaScript, Perl, PHP, Python, Ruby y Tcl son ejemplos populares de lenguajes de secuencias de comandos. Los programas escritos en lenguajes de secuencias de comandos son a menudo más cortos que los programas equivalentes escritos en lenguajes como C.

Lenguaje máquina vs. lenguaje de alto nivel



- IL (Instruction List) es el alfabeto de un lenguaje máquina.



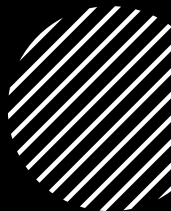
- Los lenguajes de alto nivel son más complejos que el lenguaje máquina.



- Permiten a los humanos escribir programas complejos en un formato legible.



1.1.5 Compilación vs. Interpretación



- Compilación: El programa fuente se traduce una vez a código máquina.



- Interpretación: El programa fuente se traduce cada vez que se ejecuta.



- Ventajas y desventajas de ambos modelos.

	Compilación	Interpretación
Ventajas	<ul style="list-style-type: none"> ✓ la ejecución del código traducido suele ser más rápida; ✓ solo el usuario debe tener el compilador; el usuario final puede usar el código sin él; ✓ el código traducido se almacena usando lenguaje máquina; como es muy difícil de entender, es probable que tus propios inventos y trucos de programación sigan siendo tu secreto. 	<ul style="list-style-type: none"> ✓ puedes ejecutar el código tan pronto como lo completes; no hay fases adicionales de traducción; ✓ el código se almacena usando un lenguaje de programación, no un lenguaje máquina; esto significa que se puede ejecutar en computadoras que usan diferentes lenguajes máquina; no compila tu código por separado para cada arquitectura diferente.
Desventajas	<ul style="list-style-type: none"> ✗ la compilación en sí puede ser un proceso que consume mucho tiempo; es posible que no puedas ejecutar su código inmediatamente después de realizar una modificación; ✗ debes tener tantos compiladores como plataformas de hardware donde desees que se ejecute tu código. 	<ul style="list-style-type: none"> ✗ no esperes que la interpretación acelere tu código a alta velocidad: tu código compartirá el poder de la computadora con el intérprete, por lo que no puede ser realmente rápido; ✗ tanto tu como el usuario final deben tener el intérprete para ejecutar tu código.

¿Qué hace el intérprete?



- El intérprete lee y ejecuta el código fuente línea por línea.



- Verifica errores antes de ejecutar cada línea.



- Si encuentra un error, informa dónde y qué lo causó.



- Los mensajes de error pueden ser engañosos; el problema puede estar antes del lugar del error.

Compilación vs. Interpretación: Conclusiones

- No hay una respuesta obvia sobre cuál es mejor.

- Ambos modelos tienen sus ventajas y desventajas.

- La elección depende de las necesidades del proyecto y las preferencias del programador.

Ejercicio 1.3.1: Indique cuál de los siguientes términos:

- | | | |
|-------------------------|---------------------------|--------------------------|
| a) imperativo | b) declarativo | c) von Neumann |
| d) orientado a objetos | e) funcional | f) de tercera generación |
| g) de cuarta generación | h) secuencias de comandos | |

se aplican a los siguientes lenguajes:

- | | | | | |
|---------|--------|----------|------------|---------|
| 1) C | 2) C++ | 3) Cobol | 4) Fortran | 5) Java |
| 6) Lisp | 7) ML | 8) Perl | 9) Python | 10) VB. |