



Curso de
Introducción al
Pensamiento
Computacional
con Python

David Aroesti

Objetivos

- Aprender a resolver problemas de manera computacional
- Entender los puntos en común entre todos los lenguajes de programación
- Desarrollar las bases para una carrera en Computer Science

Módulo 1

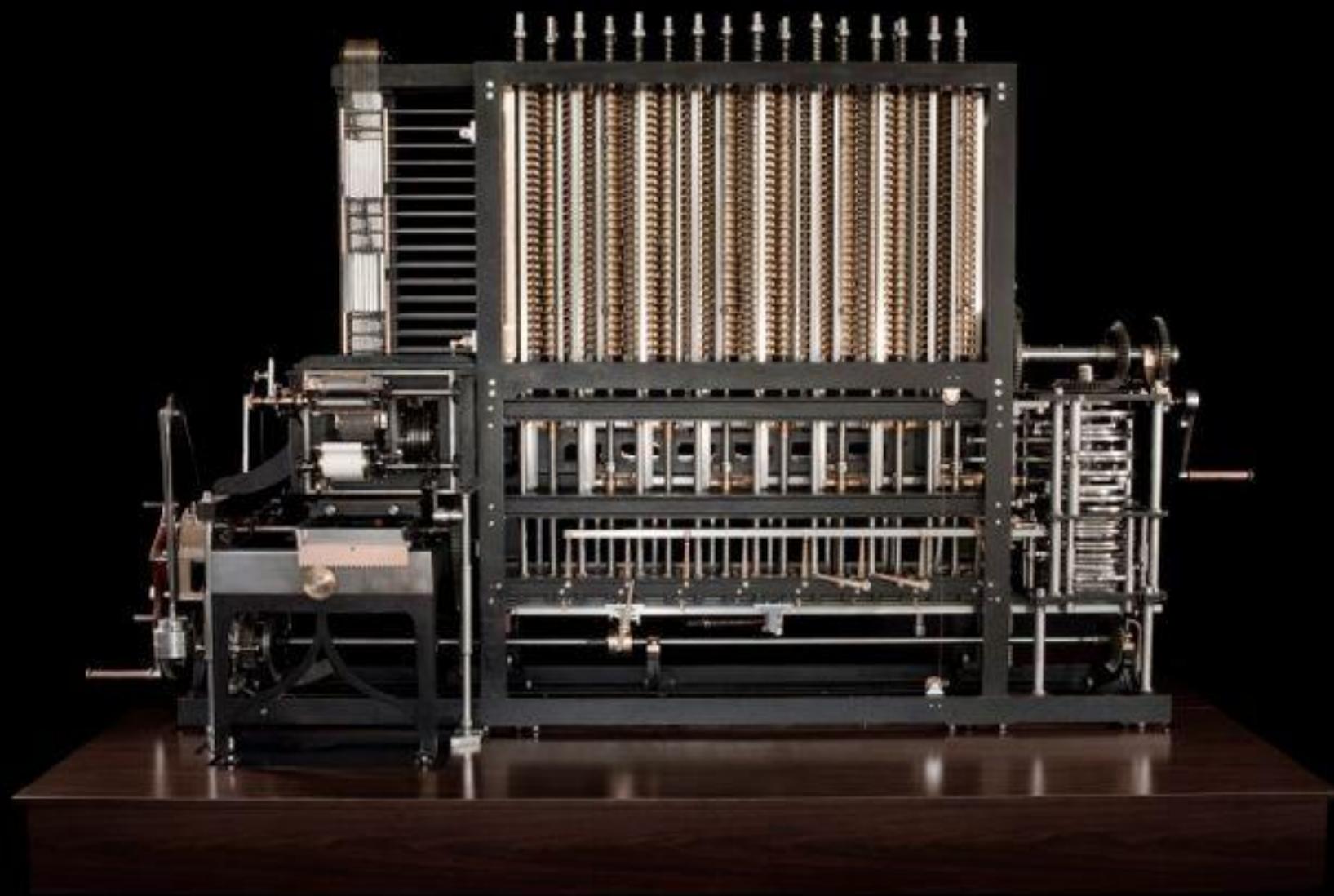
Introducción a la serie

Introducción al pensamiento computacional

Módulo 1





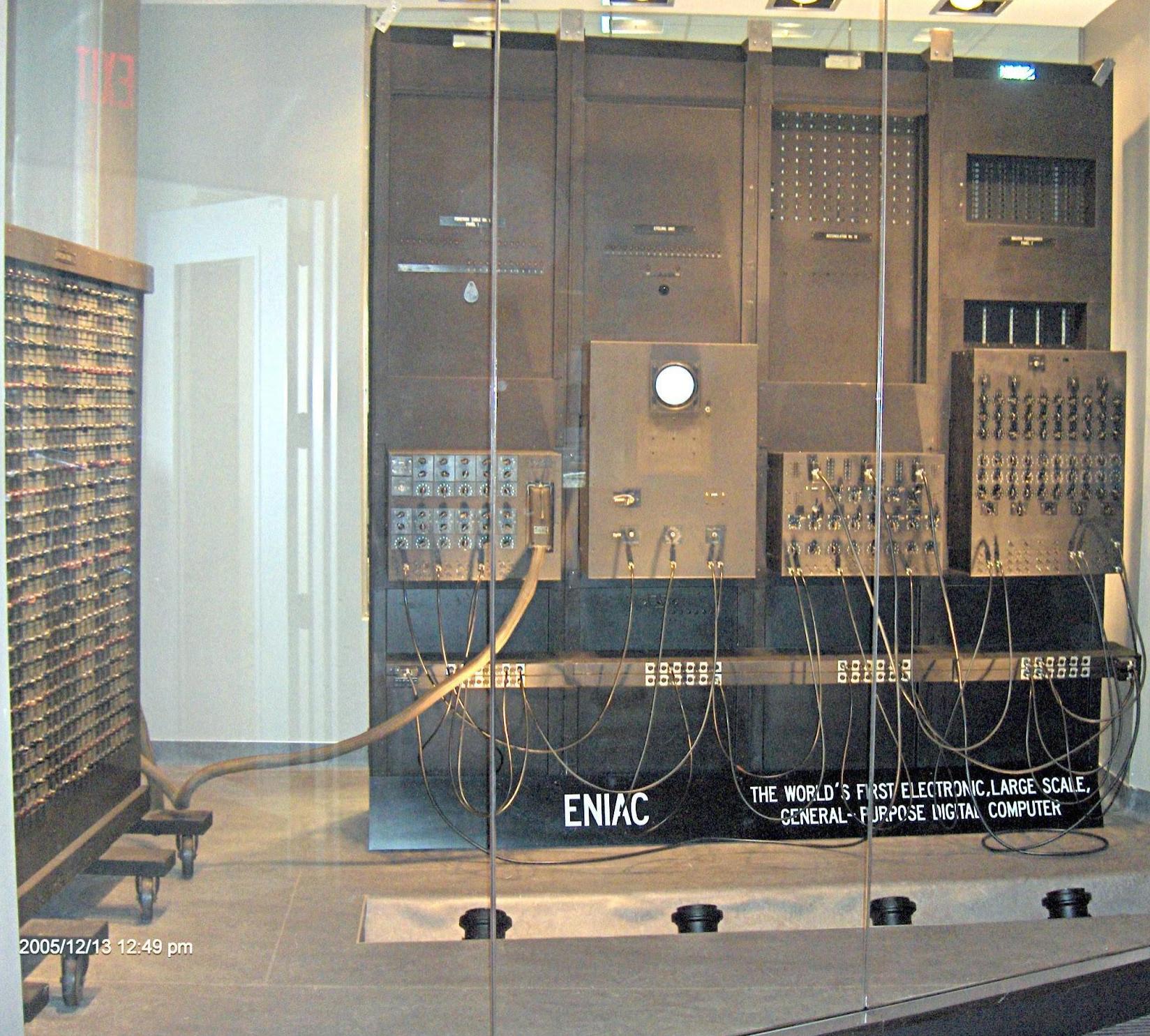






FRIDEN
Great Quality



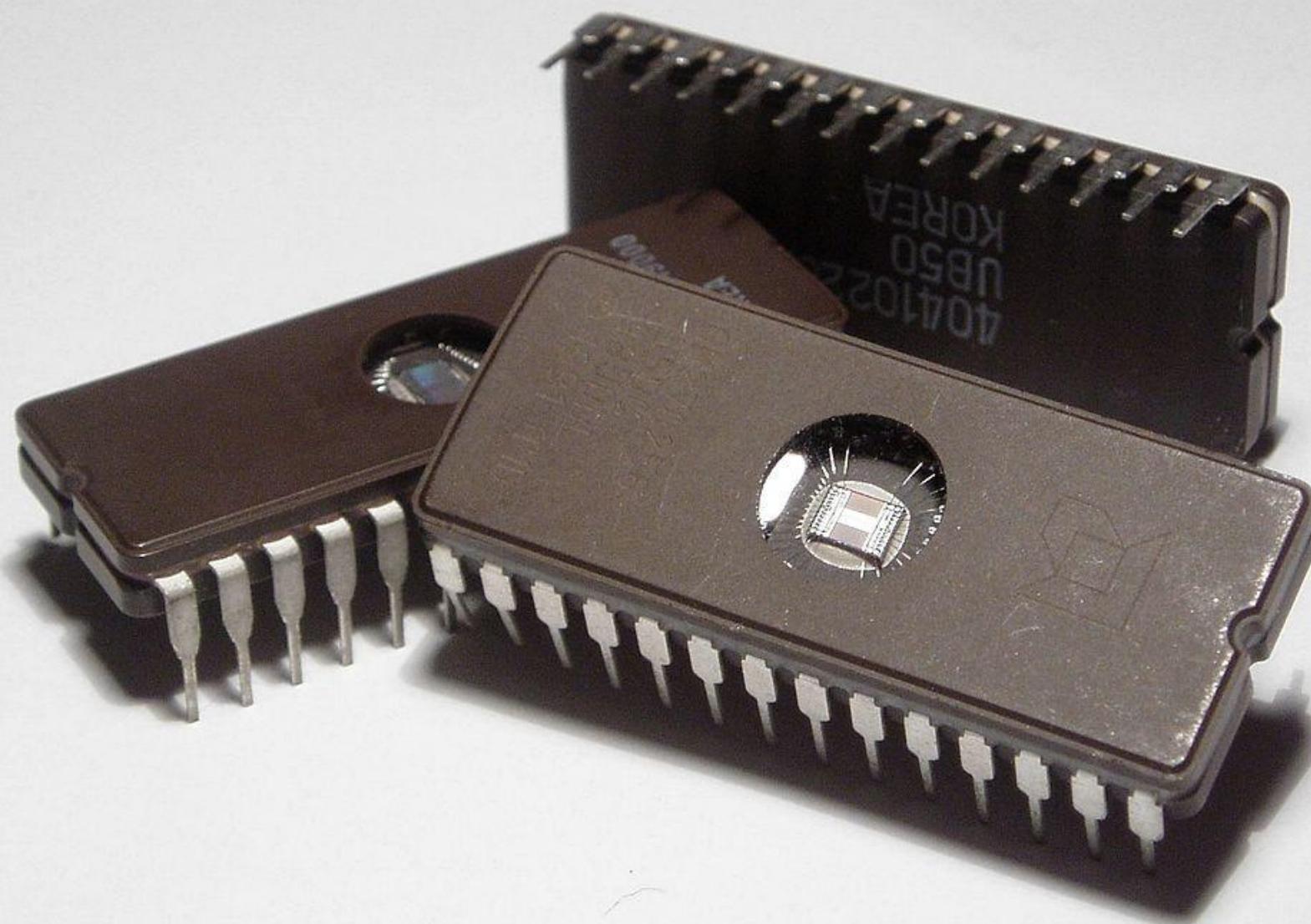


2005/12/13 12:49 pm

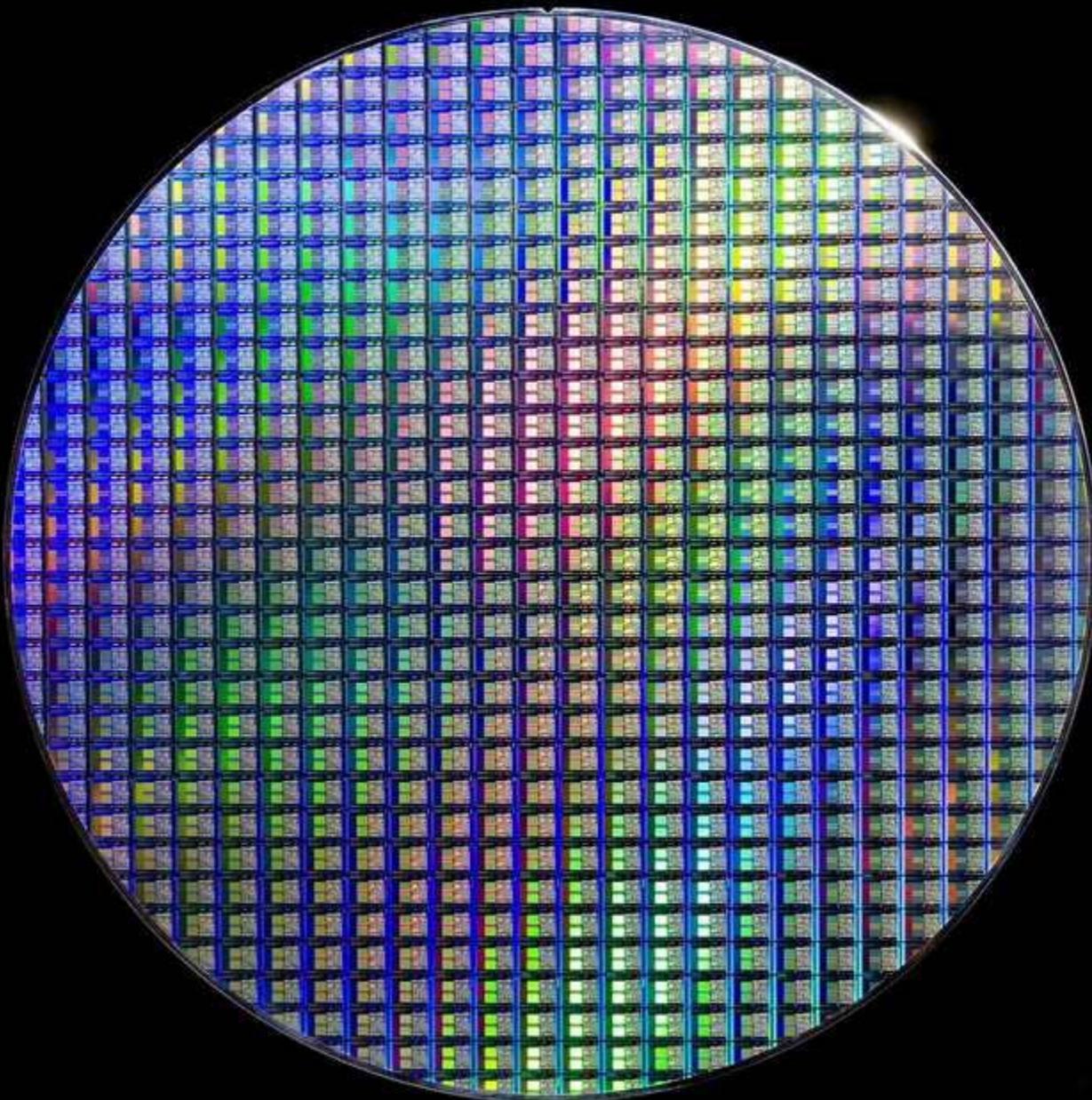
Penn
Engineering

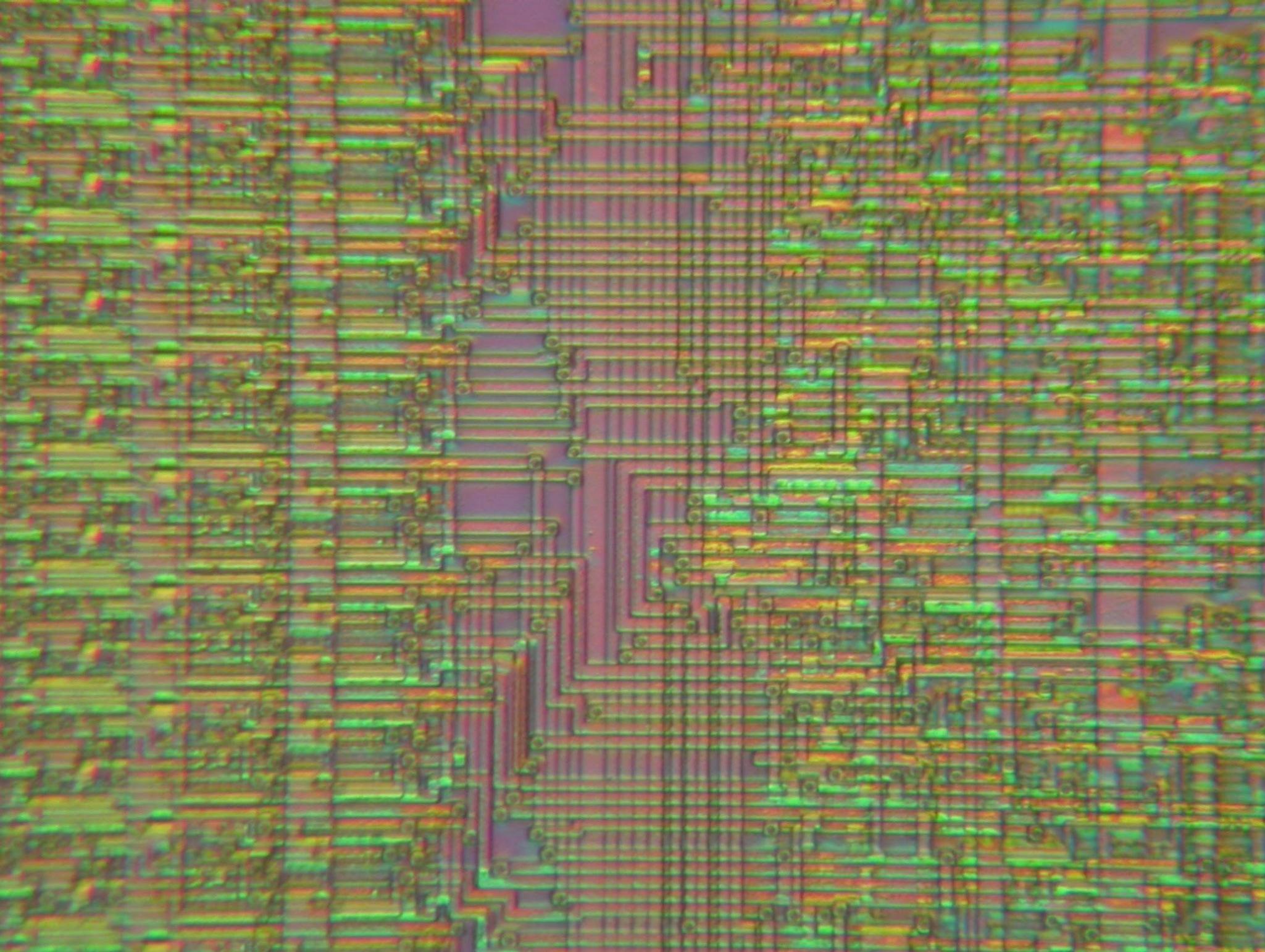


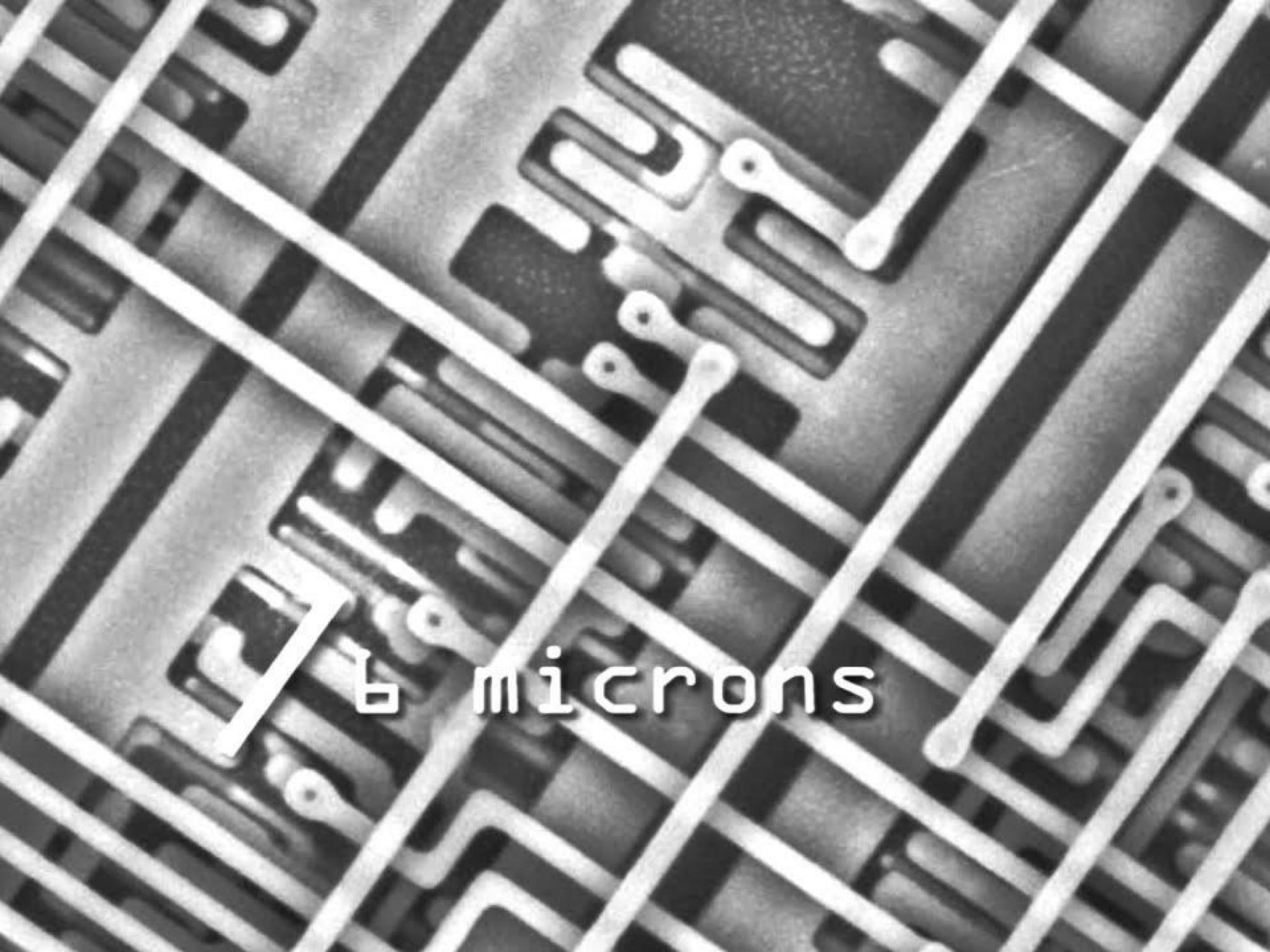






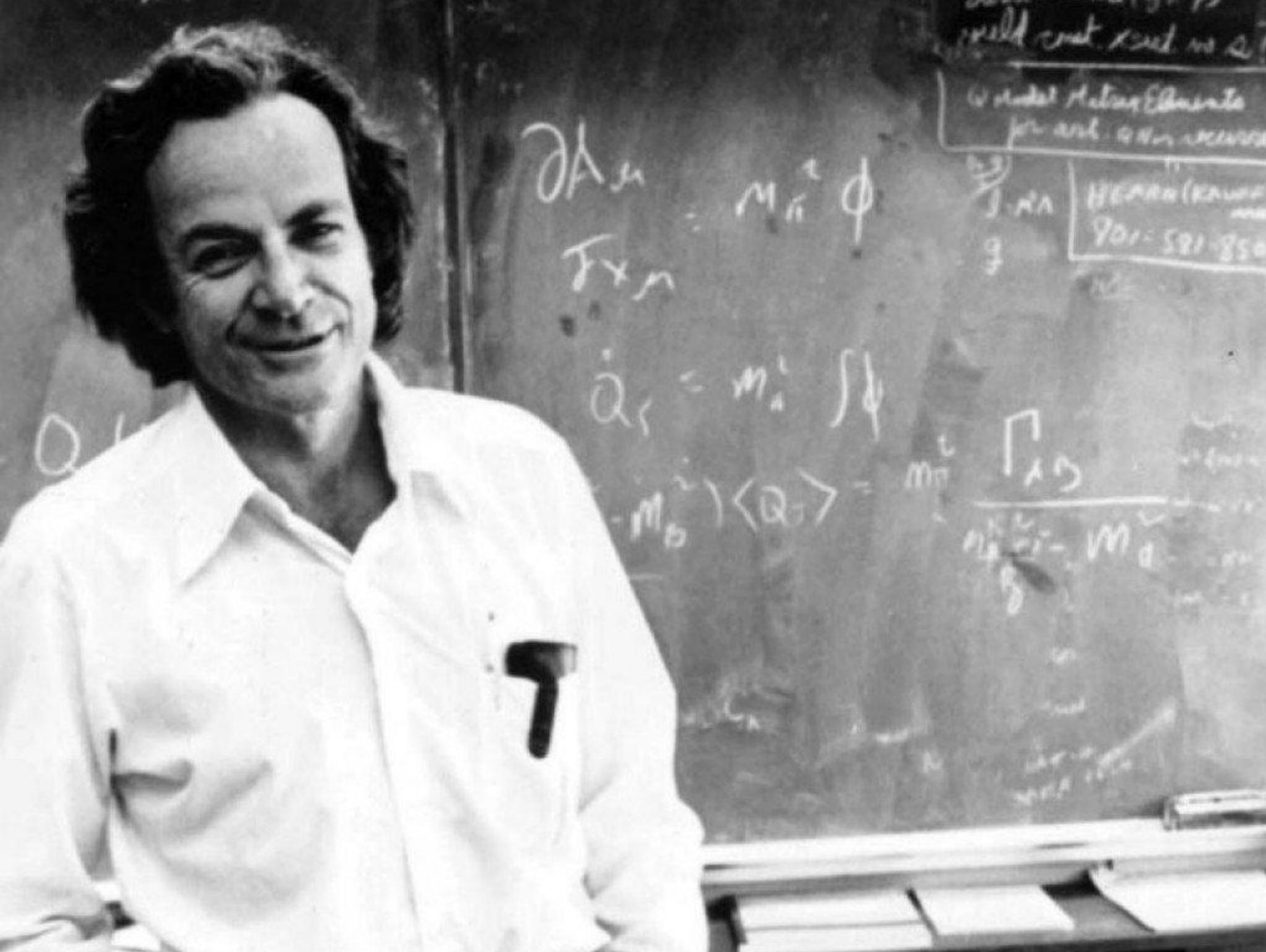


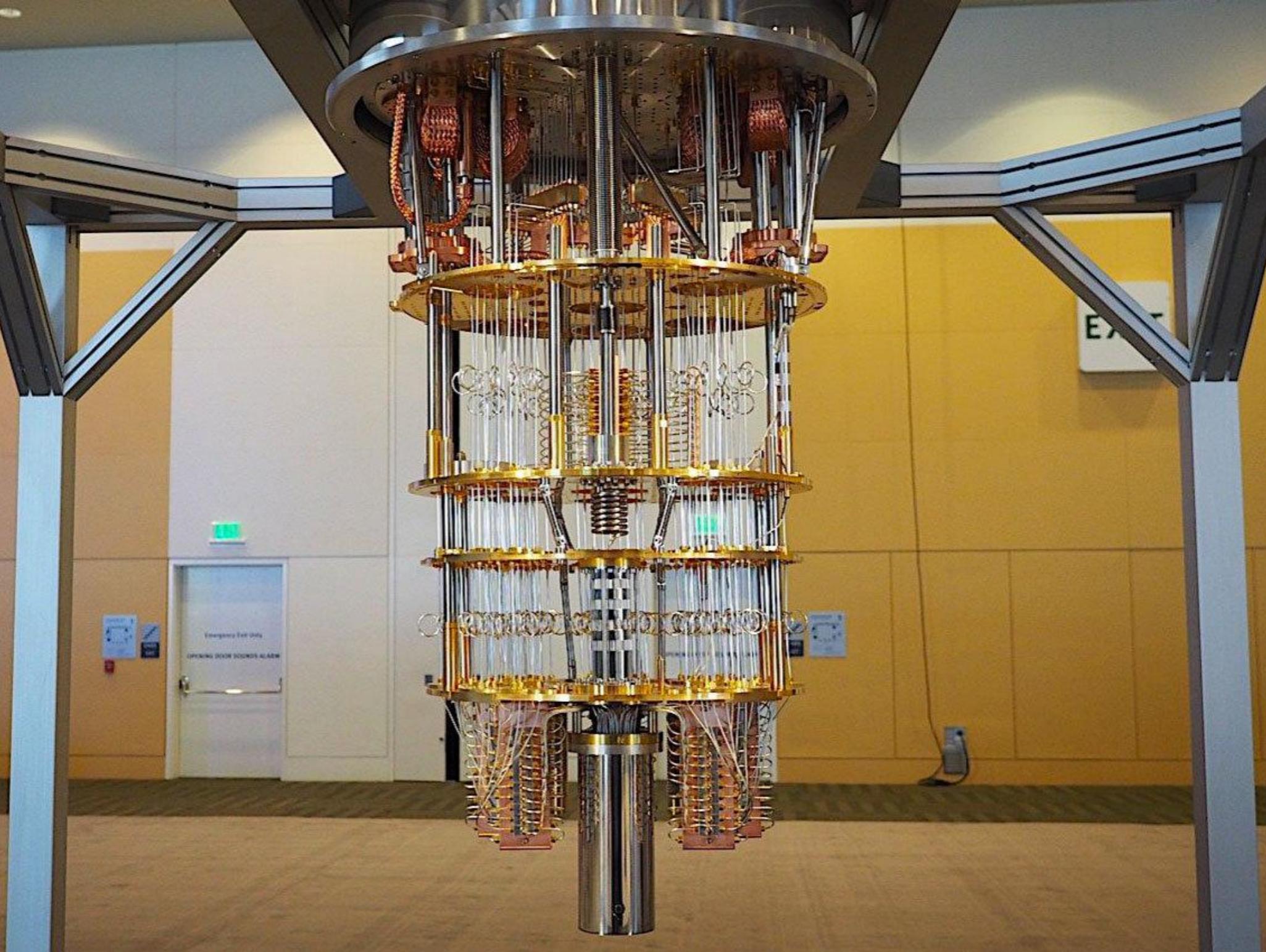


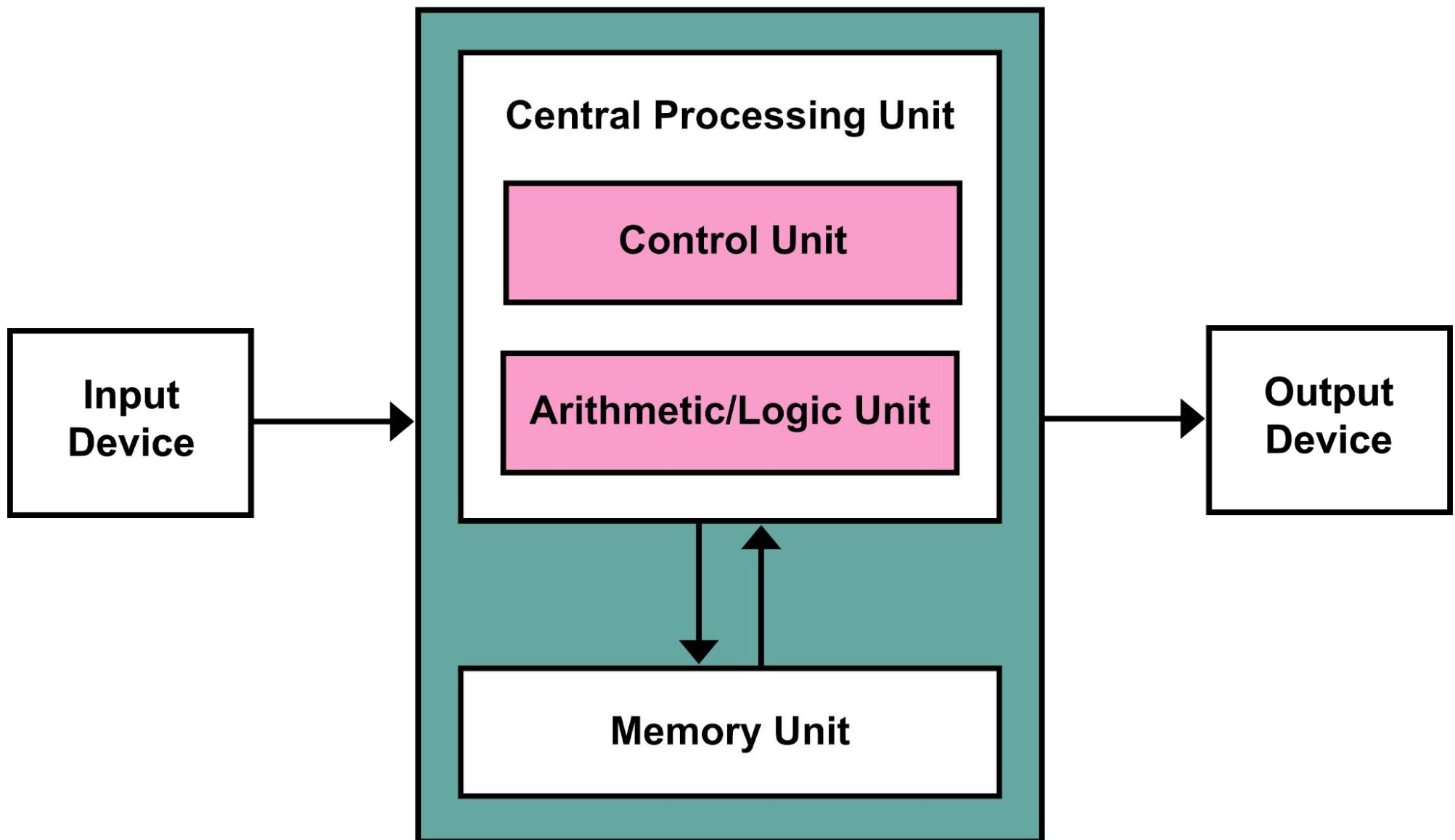


16 microns









Cómputo y computadoras

- Las computadoras hacen dos cosas: hacen cálculos y recuerdan el resultado de dichos cálculos.
- Por la mayoría de la historia humana, estábamos limitados por la velocidad del cerebro y la mano.
- Aún con las computadoras modernas existen problemas que no podemos resolver.

Introducción a los lenguajes de programación

Módulo 1

¿Cómo dar instrucciones?

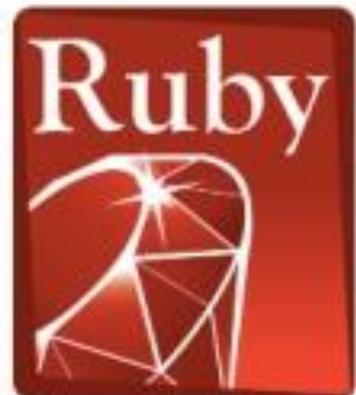
- Conocimiento declarativo vs imperativo
- Algoritmos

“

Un algoritmo es una lista finita de instrucciones que describen un cómputo, que cuando se ejecuta con ciertas entradas (inputs) ejecuta pasos intermedios para llegar a un resultado (output).

”

John V. Guttag



C#

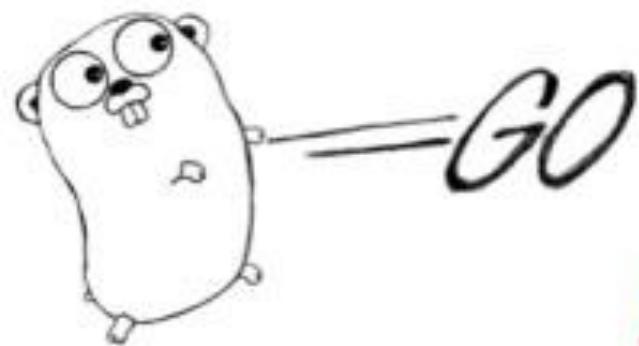


Objective-C

C++



Perl



JavaScript



Visual Basic



Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 et seq.)

| Number of Operation | Nature of Operations | Variables acted upon. | Variables receiving results. | Indication of change in the value on my Variable. | Statement of Results. | Data | | Working Variables. | | | | | | | | | | Result Variables | | |
|---|----------------------|--|---|--|--|---------------|---------------|--------------------|---------------|---------------|---------------|---------------|---------------|---------------|------------------|------------------|------------------|------------------|------------------|------------------|
| | | | | | | IV_1 | IV_2 | IV_3 | IV_4 | IV_5 | IV_6 | IV_7 | IV_8 | IV_9 | IV_{10} | IV_{11} | IV_{12} | IV_{13} | IV_{14} | |
| | | | | | | 1 | 2 | n | 1 | 2 | n | 2n | 2n | 2n | 2n | 2n | 2n | 2n | 2n | IV_{17} |
| 1 | X | $\text{IV}_2 \times \text{IV}_3$ | $\text{IV}_4 \text{ } \text{IV}_5 \text{ } \text{IV}_6$ | $\begin{cases} \text{IV}_2 = \text{IV}_3 \\ \text{IV}_3 = \text{IV}_2 \end{cases}$ | $-2n$ | | | | | | | | | | | | | | | |
| 2 | - | $\text{IV}_4 - \text{IV}_3$ | IV_4 | $\begin{cases} \text{IV}_4 = \text{IV}_3 \\ \text{IV}_3 = \text{IV}_4 \end{cases}$ | $-2n - 1$ | | | | 1 | | | $2n - 1$ | | | | | | | | |
| 3 | + | $\text{IV}_3 + \text{IV}_2$ | IV_3 | $\begin{cases} \text{IV}_3 = \text{IV}_2 \\ \text{IV}_2 = \text{IV}_3 \end{cases}$ | $-2n + 1$ | | | | 1 | | | | $2n + 1$ | | | | | | | |
| 4 | + | $\text{IV}_3 + 2\text{IV}_4$ | IV_{11} | $\begin{cases} \text{IV}_3 = \text{IV}_4 \\ \text{IV}_4 = \text{IV}_3 \end{cases}$ | $-2n - 1$ | | | | | | | | | | | | | | | |
| 5 | + | $\text{IV}_{12} + \text{IV}_3$ | IV_{11} | $\begin{cases} \text{IV}_{12} = \text{IV}_3 \\ \text{IV}_3 = \text{IV}_{12} \end{cases}$ | $1 - 2n - 1$ | | | | | | | | | | | | | | | |
| 6 | - | $\text{IV}_{12} - \text{IV}_{13}$ | IV_{12} | $\begin{cases} \text{IV}_{12} = \text{IV}_{13} \\ \text{IV}_{13} = \text{IV}_{12} \end{cases}$ | $-1 - 2n - 1$ | | | | | | | | | | | | | | | |
| 7 | - | $\text{IV}_3 - \text{IV}_1$ | IV_{30} | $\begin{cases} \text{IV}_3 = \text{IV}_1 \\ \text{IV}_1 = \text{IV}_3 \end{cases}$ | $-n - 1 (= 3)$ | | | | 1 | | | | | | | | | | | |
| 8 | + | $\text{IV}_3 + \text{IV}_7$ | IV_7 | $\begin{cases} \text{IV}_3 = \text{IV}_7 \\ \text{IV}_7 = \text{IV}_3 \end{cases}$ | $-2 + 0 = 2$ | | | | | 2 | | | | | | | | | | |
| 9 | + | $\text{IV}_8 + \text{IV}_7$ | IV_{11} | $\begin{cases} \text{IV}_8 = \text{IV}_7 \\ \text{IV}_7 = \text{IV}_8 \end{cases}$ | $\frac{2n}{2} = A_1$ | | | | | | | | | | | | | | | |
| 10 | X | $\text{IV}_{12} \times \text{IV}_{13}$ | IV_{12} | $\begin{cases} \text{IV}_{12} = \text{IV}_{13} \\ \text{IV}_{13} = \text{IV}_{12} \end{cases}$ | $B_1 + \frac{2n}{2} = B_1 A_1$ | | | | | | | | | | | | | | | |
| 11 | + | $\text{IV}_{12} + \text{IV}_{13}$ | IV_{13} | $\begin{cases} \text{IV}_{12} = \text{IV}_{13} \\ \text{IV}_{13} = \text{IV}_{12} \end{cases}$ | $-1 - 2n - 1 + B_1 \cdot \frac{2n}{2}$ | | | | | | | | | | | | | | | |
| 12 | - | $\text{IV}_{16} - \text{IV}_1$ | IV_{16} | $\begin{cases} \text{IV}_{16} = \text{IV}_{10} \\ \text{IV}_1 = \text{IV}_1 \end{cases}$ | $n - 2 (= 2)$ | | | | 1 | | | | | | | | | | | |
| 13 | - | $\text{IV}_6 - \text{IV}_1$ | IV_6 | $\begin{cases} \text{IV}_6 = \text{IV}_8 \\ \text{IV}_8 = \text{IV}_6 \end{cases}$ | $-2n - 1$ | | | | | 1 | | | | | | | | | | |
| 14 | + | $\text{IV}_1 + \text{IV}_7$ | IV_7 | $\begin{cases} \text{IV}_1 = \text{IV}_7 \\ \text{IV}_7 = \text{IV}_1 \end{cases}$ | $2 + 1 = 3$ | | | | | 1 | | | | | | | | | | |
| 15 | + | $\text{IV}_8 + 2\text{IV}_7$ | IV_8 | $\begin{cases} \text{IV}_8 = \text{IV}_6 \\ \text{IV}_6 = 2\text{IV}_7 \end{cases}$ | $\frac{2n - 1}{3}$ | | | | | | | | | | | | | | | |
| 16 | X | $\text{IV}_4 \times \text{IV}_{10}$ | IV_{11} | $\begin{cases} \text{IV}_4 = \text{IV}_6 \\ \text{IV}_6 = \text{IV}_{10} \end{cases}$ | $\frac{2n - 2n - 1}{3}$ | | | | | | | | | | | | | | | |
| 17 | - | $\text{IV}_6 - \text{IV}_1$ | IV_6 | $\begin{cases} \text{IV}_6 = \text{IV}_8 \\ \text{IV}_8 = \text{IV}_6 \end{cases}$ | $-2n - 2$ | | | | | 1 | | | | | | | | | | |
| 18 | + | $\text{IV}_1 + \text{IV}_7$ | IV_7 | $\begin{cases} \text{IV}_1 = \text{IV}_7 \\ \text{IV}_7 = \text{IV}_1 \end{cases}$ | $-3 + 1 = 4$ | | | | | 1 | | | | | | | | | | |
| 19 | + | $\text{IV}_6 + \text{IV}_7$ | IV_9 | $\begin{cases} \text{IV}_6 = \text{IV}_8 \\ \text{IV}_8 = 2\text{IV}_7 \end{cases}$ | $\frac{2n - 2}{4}$ | | | | | | | | | | | | | | | |
| 20 | X | $\text{IV}_9 \times \text{IV}_{10}$ | IV_{11} | $\begin{cases} \text{IV}_9 = \text{IV}_8 \\ \text{IV}_8 = \text{IV}_{10} \end{cases}$ | $\frac{2n - 2n - 1}{3} \cdot \frac{2n - 2}{4} = A_2$ | | | | | | | | | | | | | | | |
| 21 | X | $\text{IV}_{10} \times \text{IV}_{11}$ | IV_{30} | $\begin{cases} \text{IV}_{10} = \text{IV}_{12} \\ \text{IV}_{12} = \text{IV}_{11} \end{cases}$ | $B_2 \cdot \frac{2n - 2n - 1}{3} \cdot \frac{2n - 2}{3} = B_2 A_2$ | | | | | | | | | | | | | | | |
| 22 | + | $\text{IV}_{12} + \text{IV}_{13}$ | IV_{13} | $\begin{cases} \text{IV}_{12} = \text{IV}_{10} \\ \text{IV}_{10} = \text{IV}_{13} \end{cases}$ | $A_2 + B_1 A_1 + B_2 A_2$ | | | | | | | | | | | | | | | |
| 23 | - | $\text{IV}_{16} - \text{IV}_1$ | IV_{30} | $\begin{cases} \text{IV}_{16} = \text{IV}_{10} \\ \text{IV}_{10} = \text{IV}_1 \end{cases}$ | $n - 3 (= 1)$ | | | | 1 | | | | | | | | | | | |
| Here follows a repetition of Operations thirteen to twenty-three. | | | | | | | | | | | | | | | | | | | | |
| 24 | + | $\text{IV}_{10} + \text{IV}_{12}$ | IV_{30} | $\begin{cases} \text{IV}_{10} = \text{IV}_{12} \\ \text{IV}_{12} = \text{IV}_{10} \end{cases}$ | $= B_2$ | | | | | | | | | | | | | | | |
| 25 | + | $\text{IV}_1 + \text{IV}_2$ | IV_2 | $\begin{cases} \text{IV}_1 = \text{IV}_2 \\ \text{IV}_2 = \text{IV}_1 \end{cases}$ | $n + 1 = 4 + 1 = 5$ | 1 | | | | | | | | | | | | | | |
| | | | | by a Variable-card. | | | | | | | | | | | | | | | | |
| | | | | $\text{IV}_7 = \text{IV}_2$ | | | | | | | | | | | | | | | | |

 IV₁ IV₂ IV₃
 1 0 0
 n decimal fraction.
 B₁
 B₂

 IV₄ IV₅ IV₆
 0 0 0
 n decimal fraction.
 B₃

 IV₇ IV₈ IV₉
 0 0 0
 n decimal fraction.
 B₄

 IV₁₀ IV₁₁ IV₁₂
 0 0 0
 n decimal fraction.
 B₅

 IV₁₃ IV₁₄ IV₁₅
 0 0 0
 n decimal fraction.
 B₆

 IV₁₆ IV₁₇ IV₁₈
 0 0 0
 n decimal fraction.
 B₇



F0 FE 14 04 1C 70 04 A0 D0 80 EF 00 70 D4 B2 C0 BB 80 05
FB C0 50 D8 F7 00 00 BB EF E0 F0 D1 00 0E B0 D4 50 00 EE
E2 B6 B4 FB 08 FF B1 A0 40 F2 EE 50 E0 04 4C 0F 03 1C F2
03 D1 EE 70 A0 B6 A2 E0 B9 20 04 FD 03 AD B2 01 C2 C1 E8
E4 03 A2 20 0F C9 C4 BB C0 C2 0A EB C0 A5 EE F8 20 EF 05
EB C0 80 EF BB F8 05 A4 30 F4 FE 14 80 FF B1 A0 D1 80 EF
14 80 E0 F0 50 E0 AE DA B2 FB C0 90 A0 B6 A2 0F EF E3 F0
5F A0 F2 EE 03 BC F2 A5 EE E2 B6 B4 0F C9 C4 BB C0 F2 EE
B4 0F 20 04 C2 C4 E8 F7 B1 03 D1 EE BB 70 0E DB 0B 20 04
EC BE C7 06 A0 EF D5 B6 A2 E4 05 A2 30 00 E5 03 C0 C2 50
F7 00 B2 01 C2 C5 04 0F 03 A0 70 A0 B6 A2 0A B6 A2 E0 BB
B0 DF CE F8 20 EF B2 01 C2 0F 20 0F C9 C4 0F C9 C4 BB C0
A0 B6 1B A5 DC A3 EE F8 20 BB EF CB D0 05 EF BA C0 C2 C2
0F C9 A7 F0 EF E0 B1 AC D9 2A F0 70 00 EE C1 F4 40 FA F0
BB 80 C4 BB D0 F2 DA B2 FB 03 1C F2 A5 E9 EF D3 B2 FB FB
80 EF A2 20 0F EE 70 A0 04 C0 DA B2 B1 EF CE C4 F2 A5 EE
E0 F0 F2 B7 03 A6 E0 B1 DA B2 C1 02 A0 04 C8 05 E8 FF B1
FF B1 E9 EB B6 A2 E4 BB A5 EC EF DA E0 BC 00 ED E5 B6 A2
B6 A2 02 05 0F 03 00 5F F0 B1 D2 C4 BB C0 A2 A8 40 0F 03
C9 A4 C0 B6 01 C2 0F 03 B6 A9 EF C9 D4 40 C4 0F B2 01 C2
C0 C2 50 0B CE F8 20 EF B2 10 BC 20 07 FD E2 03 CF EE F8

IS YOURS?

Storage data management for process control, plant automation and supervisory control systems. Storage and retrieval of data from databases, historians, and distributed and federated databases. It gives the operator the full functionality for the management of the information systems across the distributed links in the system, reducing and updating data automatically and periodically.

apack

DATA
MANAGEMENT

O

magtape

```
def main():
    print('hello world!')
```

```
if __name__ == '__main__':
    main()
```





Programación

- Turing completeness
- Los lenguajes de programación modernos dan primitivos que son más convenientes que los primitivos de Turing.

Lenguajes

- **Sintaxis**
Define la secuencia de símbolos que está bien formada.
- **Semántica estática**
Define qué enunciados con sintaxis correcta tienen significado
- **Semántica**
Define el significado. En los lenguajes de programación sólo hay un significado.

Módulo 2

Introducción a Python

Elementos básicos de Python

Módulo 2

Lenguajes de programación

- Bajo nivel vs. alto nivel
- General vs. dominio específico
- Interpretado vs. compilado

Elementos básicos de Python

```
# <literales> = 1, 'abc', 2.0, True
# <operadores> = + / * % ** = ==
# <literal> <operador> <literal>

>>> 1 + 2
>>> 1 3.0 # error sintáctico
>>> 5 / 'Platzi' # error semántico estático
>>> 5 * 'Platzi'

# statement o enunciado
>>> print('Hello, Platzi!')
```

Objetos, expresiones y tipos numéricos

Objetos

- Objetos
- Tipos
- Escalares vs. no escalares

```
<objeto> <operador> <objeto> # expresión  
">>>> <valor>
```

```
>>> 'Platzi' + 'Rocks!'  
PlatziRocks!
```

```
>>> 2 + 2  
4
```

```
>>> persona + persona  
pareja
```

```
>>> my_int = 1
>>> my_float = 1.0
>>> my_bool = False
>>> my_none = None

>>> type(my_int)
>>> type(my_float)
>>> type(my_bool)
>>> type(my_none)
```

```
>>> 1 + 2  
>>> 2 - 5  
>>> 2.0 * 3  
>>> 6 // 2  
>>> 6 // 4  
>>> 6 / 4  
>>> 7 % 2  
>>> 2**3
```

Asignación de variables

a = 2

x = 4

z = (a * x) / 2

```
base = 2
```

```
altura = 4
```

```
area = (base * altura) / 2
```

```
>>> my_var = 'Hello, Platzi'  
>>> print(my_var)  
Hello, Platzi
```

```
>>> my_var = 3  
>>> print(my_var)  
3
```

my_var



| | |
|--------|-----------------|
| ... | ... |
| 0x0001 | 'Hello, Platzi' |
| 0x0002 | 3 |
| ... | ... |

my_var



| | |
|--------|-----------------|
| ... | ... |
| 0x0001 | 'Hello, Platzi' |
| 0x0002 | 3 |
| ... | ... |

Variables

- Pueden contener mayúsculas, minúsculas, números (sin comenzar con uno) y el símbolo _
- No pueden llamarse como las palabras reservadas

Variables

| | | | | |
|--------|----------|---------|----------|--------|
| False | await | else | import | pass |
| None | break | except | in | raise |
| True | class | finally | is | return |
| and | continue | for | lambda | try |
| as | def | from | nonlocal | while |
| assert | del | global | not | with |
| async | elif | if | or | yield |

Variables

- Hacen los programas más comprensibles.
- Son simplemente nombres que apuntan a un valor en memoria.
- El operador de asignación (=) asocia una variable con un valor.

Cadenas y entradas

Módulo 2

Cadenas

```
>>> '123'  
  
>>> '123' * 3  
  
>>> '123' + '456'  
  
>>> ('Hip ' * 3) + ' ' + 'hurra'  
  
>>> f'{ "Hip " * 3} hurra'
```

Cadenas (strings)

- len (longitud)
- Indexing (indexación)
- slicing (rebanadas)
 - `my_str[comienzo:fin:pasos]`

Cadenas (strings)

- Los objetos de tipo str pueden representarse con “ ” o ‘’’.
- El operador + tiene diferente significado según el tipo de dato (overloaded). Con cadenas significa concatenación.
- El operador * es el operador de repetición con cadenas.
- Las cadenas son inmutables.

Entradas

Entradas (inputs)

- Python tiene la función `input` para recibir datos del usuario del programa.
- `Input` siempre regresa cadenas, por lo que si queremos utilizar otro tipo, tenemos que hacer *type casting*.

Programas ramificados

Módulo 2

```
>>> 2 == 3  
>>> 2 != 3  
>>> 2 > 3  
>>> 2 < 3  
>>> 2 <= 3  
>>> 2 >= 3
```

```
>>> True and True
```

```
>>> False or True
```

```
>>> not True
```

| A | B | A and B | A or B | Not B |
|-------|-------|---------|--------|-------|
| True | True | True | True | False |
| True | False | False | True | True |
| False | True | False | True | |
| False | False | False | False | |

```
if <condition>:  
    <expresion>
```

```
if 3 > 2:  
    print('3 es mayor que 2')
```

```
if <condition>:  
    <expresion>  
else:  
    <expresion>
```

```
if 5 <= 10:  
    ...  
else:  
    print('5 no es mayor o igual que 10')
```

```
if <condition>:  
    <expresion>  
elif <condition>:  
    <expresion>  
else:  
    <expresion>
```

```
if 4 > 5:  
    ...  
elif 4 < 5:  
    print('4 es menor que 5')  
  
else:  
    ...
```

Iteraciones

Módulo 2

Iteraciones (loops)

- La mayoría de las tareas computacionales no se pueden lograr con ramificaciones.
- Cuando queremos que un programa haga lo mismo varias veces, utilizamos iteraciones.
- Se pueden escribir iteraciones dentro de iteraciones.
- Podemos utilizar *break* para salir anticipadamente de una iteración.
- Tener cuidado de iteraciones infinitas.

```
while <condicion>:  
    <expresion>
```

```
while <otra condicion>:  
    <otra expresion>
```

Módulo 3

Programas numéricos

Enumeración exhaustiva

Módulo 3

Enumeración exhaustiva

- También llamado “adivina y verifica”.
- Las computadoras actuales son muy muy rápidas.
- Uno de los primeros algoritmos que debes tratar.

Aproximación de soluciones y búsqueda binaria

Módulo 3

Aproximación de soluciones

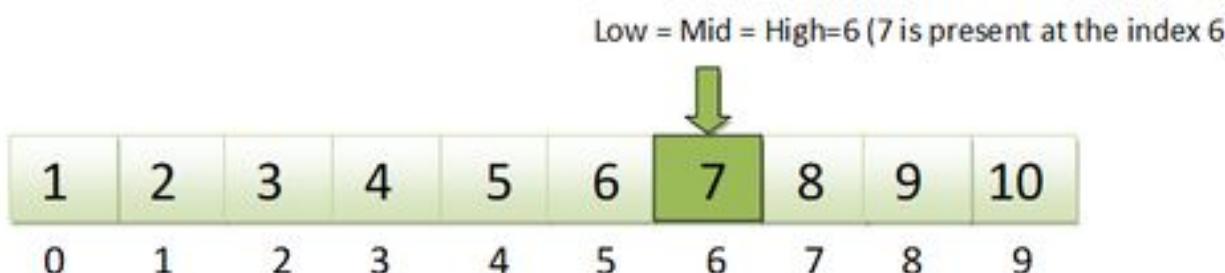
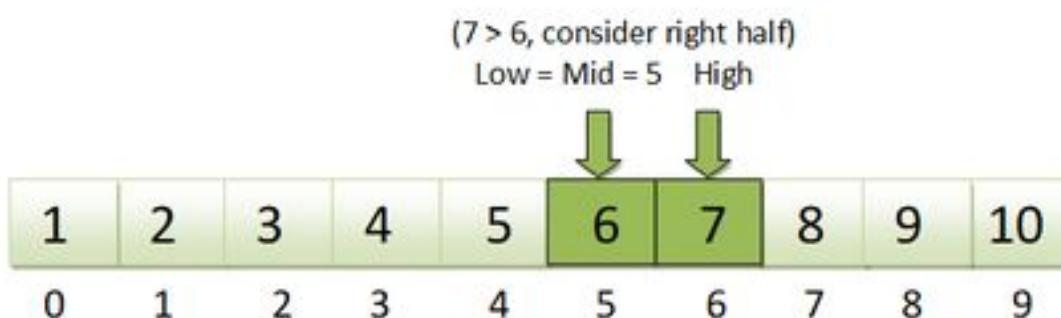
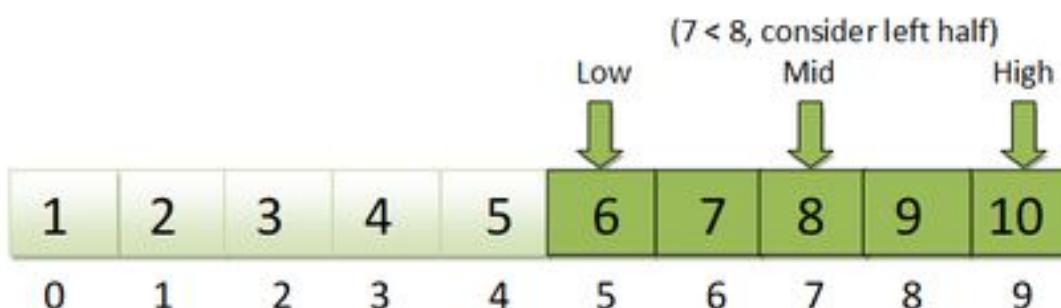
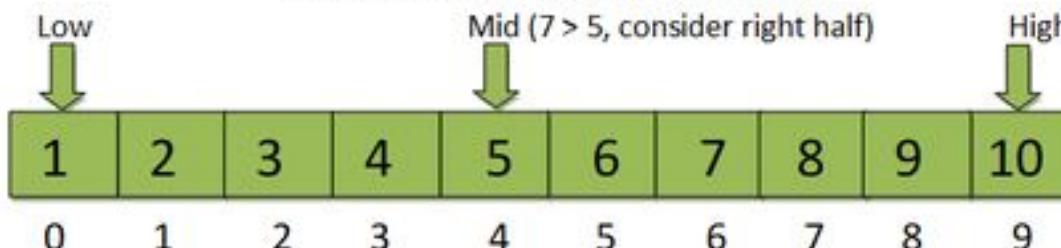
- Similar a enumeración exhaustiva, pero no necesita una respuesta exacta.
- Podemos aproximar soluciones con un margen de error que llamaremos epsilon.

Búsqueda binaria

- Cuando la respuesta se encuentra en un conjunto ordenado, podemos utilizar búsqueda binaria
- Es altamente eficiente, pues corta el espacio de búsqueda en dos por cada iteración

Binary Search

Search the number 7 in the array



Módulo 4

Funciones, alcance y abstracción

Funciones y alcance de las funciones

Módulo 4

Abstracción



Decomposición

- Permite dividir el código en componentes que colaboran con un fin en común.
- Se puede pensar como mini programas dentro de un programa mayor

Definición de funciones

```
def <nombre>(<parametros>):  
    <cuerpo>  
    return <expresion>
```

```
def suma(a, b):  
    total = a + b  
    return total
```

```
suma(2, 3)
```

Argumentos de Keyword y valores por defecto

```
def nombre_completo(nombre, apellido, inverso=False):  
    if inverso:  
        return f'{apellido} {nombre}'  
    else:  
        return f'{nombre} {apellido}'
```

```
nombre_completo('David', 'Aroesti')
```

```
nombre_completo('David', 'Aroesti', inverso=True)
```

```
nombre_completo(apellido='Aroesti', nombre='David')
```

Alcance

```
def func1(un_arg, una_func):  
    def func2(otro_arg):  
        return otro_arg * 2
```

```
    valor = func2(un_arg)  
    return una_func(valor)
```

```
un_arg = 1
```

```
def cualquier_func(cualquier_arg):  
    return cualquier_arg + 5
```

```
func1(un_arg, cualquier_func)
```

Especificaciones del código

Módulo 4

```
def suma(a, b):  
    """Suma dos valores a y b.  
  
    param int a cualquier entero  
    param int b cualquier entero  
    returns la sumatoria de a y b  
    """  
  
    total = a + b  
  
    return total
```

Recursividad

Módulo 4

Recursividad

- **Algorítmica**
Una forma de crear soluciones utilizando el principio de “divide y vencerás.”
- **Programática**
Una técnica programática mediante la cual una función se llama a sí misma.

Factoriales

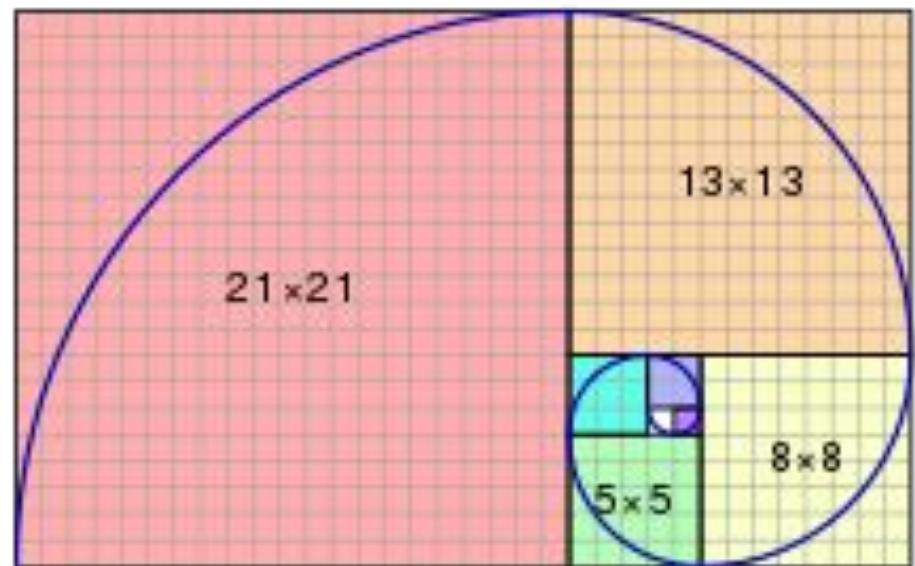
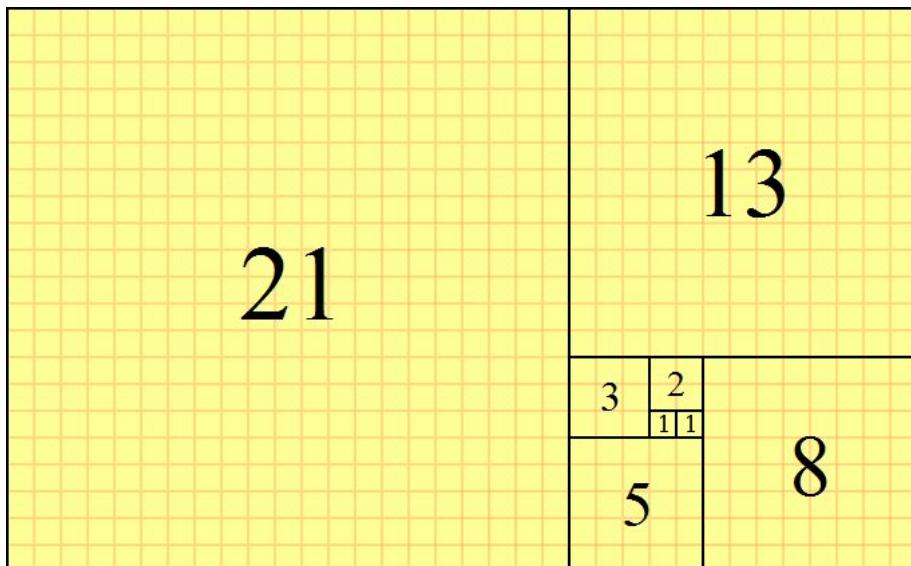
Factoriales

$$n! = \prod_{i=1}^n i.$$

$$n! = n \cdot (n - 1)!$$

Números Fibonacci

Fibonacci



Módulo 5

Tipos estructurados, mutabilidad
y funciones de alto nivel

Tuplas

Módulo 5

Tuplas

- Son secuencias inmutables de objetos.
- A diferencia de las cadenas pueden contener cualquier tipo de objeto.
- Puede utilizarse para devolver varios valores en una función.

Rangos

Módulo 5

Rangos

- Representan una secuencia de enteros.
- `range(comienzo, fin, pasos)`
- Al igual que las cadenas y las tuplas, los rangos son inmutables.
- Muy eficientes en uso de memoria y normalmente utilizados en for loops

Listas y mutabilidad

Módulo 5

Listas y mutabilidad

- Son secuencias de objetos, pero a diferencia de las tuplas, sí son mutables.
- Cuando modificas una lista, pueden existir efectos secundarios (*side effects*)
- Es posible iterar con ellas.

Listas y mutabilidad

- Para modificar una lista podemos:
 - Asignar vía índice (*my_lista[0] = 5*)
 - Utilizar los métodos de la lista (*append, pop, remove, insert, etc.*)

Clonación

Clonación

- Casi siempre es mejor clonar una lista en vez de mutarla
- Para clonar una lista podemos utilizar rebanadas (slices) o la función list

List comprehension

List comprehension

- Es una forma concisa de aplicar operaciones a los valores de una secuencia.
- También se pueden aplicar condiciones para filtrar.

Diccionarios

Módulo 5

Diccionarios

- Son como listas, pero en lugar de usar índices utilizan llaves.
- No tienen orden interno.
- Los diccionarios son mutables.
- Pueden iterarse.

Módulo 6

Pruebas y debugging

Pruebas de caja negra

Módulo 6

Pruebas de caja negra

- Se basan en la especificación de la función o el programa.
- Prueba inputs y valida outputs.
- *Unit testing* o *integration testing*.

Pruebas de caja de cristal

Módulo 6

Pruebas de caja de cristal

- Se basan en el flujo del programa.
- Prueba todos los caminos posibles de una función. Ramificaciones, bucles for y while, recursión.
- *Regression testing o mocks.*

Debugging

Módulo 6

IS YOURS?

Storage data management for process control, plant automation and supervisory control systems. Storage and retrieval of data from databases, historians, and distributed and centralized. It gives the operator the necessary data information required across the organization. It is the central information system for the plant.

apack

magtape

92

9/9

0800

Anbar started

1000

" stopped - anbar ✓

{ 1.2700 9.037847025

9.037846795 const

13' UC (033) MP - MC

~~1.9552147000~~~~2.130476715(-2)~~

(033) PRO 2

2.130476415

const

2.130676415

Relays 6-2 in 033 failed special speed test
in relay

Relay

2145

Relay 3371

1100

Started Cosine Tape (Sine check)

1525

Started Multi Adder Test.

1545

Relay #70 Panel F
(moth) in relay.

1600

First actual case of bug being found.

1700

Anbar started.

closed down.

Reglas generales

Reglas generales

- No te molestes con el debugger. Aprende a utilizar el print statement.
- Estudia los datos disponibles.
- Utiliza los datos para crear hipótesis y experimentos.
Método científico
- Ten una mente abierta. Si entendieras el programa, probablemente no habrían bugs.
- Lleva un registro de lo que has tratado, preferentemente en la forma de tests.

Diseño de experimentos

Diseño de experimentos

- Debugear es un proceso de búsqueda. Cada prueba debe acotar el espacio de búsqueda.
- Búsqueda binaria con print statements.

Errores comunes

Errores comunes

- Encuentra a los sospechosos comunes.
- En lugar de preguntarte por qué un programa no funciona, pregúntate por qué está funcionando de esta manera.
- Es posible que el bug no se encuentre donde crees que está.
- Explícale el problema a otra persona. De preferencia que no tenga contexto.
- Lleva un registro de lo que has tratado, preferentemente en la forma de tests.
- Vete a dormir.

Módulo 7

Excepciones y afirmaciones

Manejo de excepciones

Módulo 7

Manejo de excepciones

- Son muy comunes en la programación. No tienen nada de excepcional.
- Las excepciones de Python normalmente se relacionan con errores de semántica.
- Se pueden crear excepciones propias.
- Cuando una excepción no se maneja (unhandled exception), el programa termina en error.

Manejo de excepciones

- Las excepciones se manejan con los keywords: `try`, `except`, `finally`.
- Se pueden utilizar también para ramificar programas.
- No deben manejarse de manera silenciosa (por ejemplo, con `print statements`)
- Para aventar tu propia excepción utiliza el keyword `raise`

Afirmaciones

Módulo 7

Afirmaciones

- Programación defensiva
- Pueden utilizarse para verificar que los tipos sean correctos en una función
- También sirven para debuguear

```
# assert <expresion booleana>, <mensaje de error>

def primera_letra(lista_de_palabras):
    primeras_letras = [ ]

    for palabra in lista_de_palabras:
        assert type(palabra) == str, f'{palabra} no es str'
        assert len(palabra) > 0, 'No se permiten str vacios'

        primeras_letras.append(palabra[0])

    return primeras_letras
```

Cierre del curso

Conclusiones