Programación

Tema 1. Sistemas, Programación y Tipos Básicos

Javier González Villa

David Lázaro Urrutia

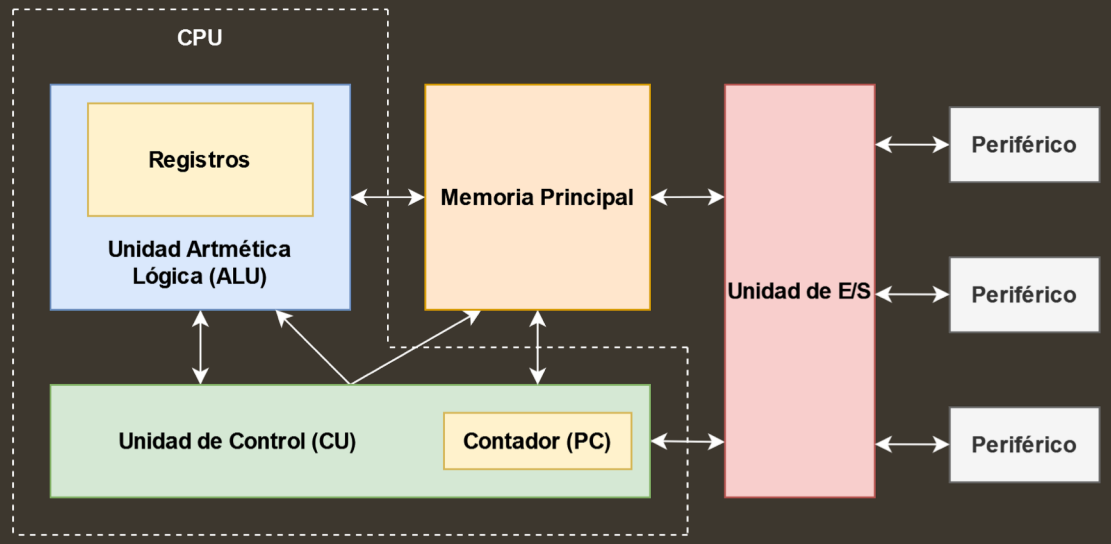
Departamento de Matemática Aplicada
y Ciencias de la Computación

Contenidos

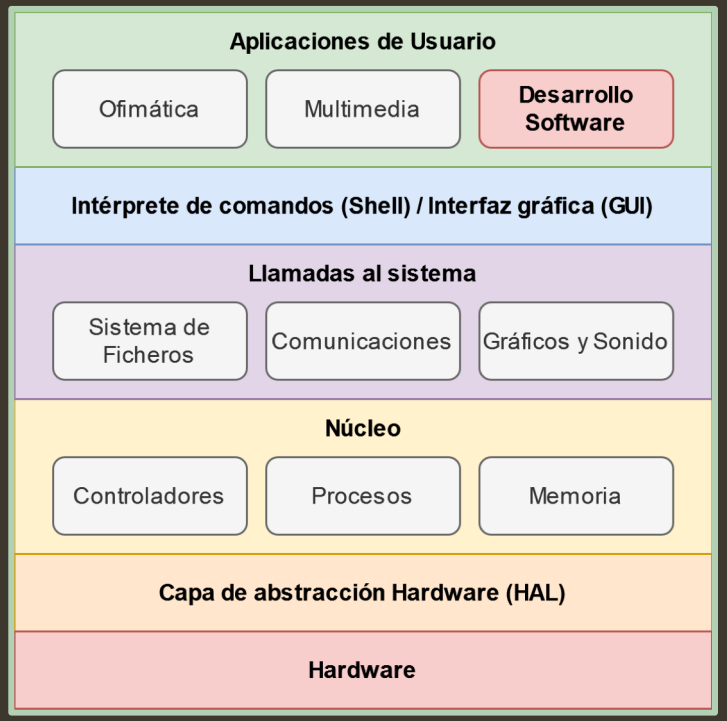
1. Sistemas Operativos
2. Lenguajes de programación
    1. Números binarios
    2. Lenguajes de bajo nivel
    3. Lenguajes de alto nivel
3. Programación en Python
    1. Clases de objetos y variables
    2. Operadores matemáticos y lógicos
    3. E/S Teclado

1. Sistemas Operativos

Sistema Operativo: *gestor de recursos hardware que provee de servicios a los programas de aplicación. Automatiza muchas rutinas haciendo de intermediario entre el usuario y la computadora.*



Arquitectura de Von Neumann



2. Lenguajes de programación

> ***Lenguaje de programación:*** *conjunto de reglas gramaticales bien definidas que proporcionan a una persona la capacidad de controlar un sistema informático a través de conjuntos de secuencias de instrucciones.*

- Una computadora es una máquina capaz de ejecutar secuencias de instrucciones almacenadas en memoria.
- Cada instrucción le transmite al computador una orden de lo que debe de hacer.
- Los lenguajes de programación, estructuran una gramática que permite definir de manera correcta y ordenada esas secuencias de instrucciones para definir órdenes más complejas, al ser prácticamente imposible que un ordenador comprenda el lenguaje natural.
- Los lenguajes de programación son sencillos para que estos puedan ser fácilmente interpretados tanto por el programador como por la computadora.

2.1. Números binarios

Sistema binario: *técnica de numeración donde solo se utilizan dos dígitos (0,1) para la realización de operaciones.*

- Es necesario utilizar estos números ya que los computadores están construidos mediante circuitos electrónicos digitales que normalmente solo manejan dos estados: voltaje alto (1) o voltaje bajo (0).

- Por ellos la realización de operaciones debe realizarse en binario así como la representación de cualquier información de mayor tamaño, concatenando secuencias de ceros y unos para su representación:

$$100101011$$
$$256 + 0 + 0 + 32 + 0 + 8 + 0 + 2 + 1$$
$$100101011 = 299$$

2.1. Números binarios

Codificación: *aplicación inyectiva c de los elementos de un conjunto finito (alfabeto fuente)* A *al alfabeto código* $\{0,1\}^n$. $c: \mathrm{A} \rightarrow \{0,1\}^n$

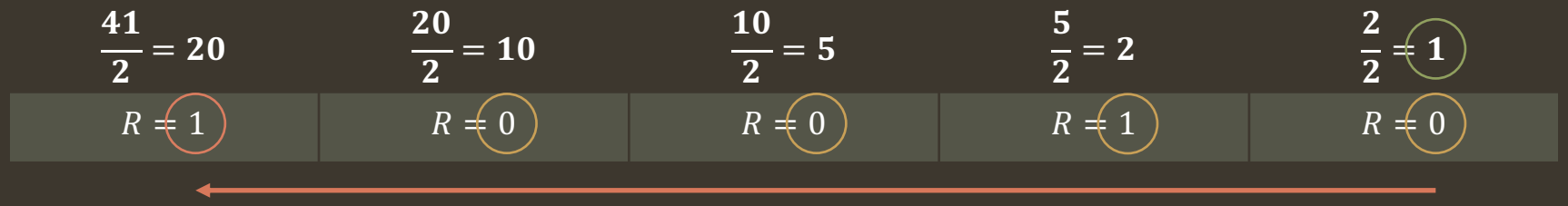
- Cada elemento del alfabeto código es denominado *bit* que se concatenan para formar conjuntos de información más largos:
  - 1 bit: 2 combinaciones (0, 1)
  - 2 bits: 4 combinaciones (00, 01, 10, 11)
  - 3 bits: 8 combinaciones (000, 001, 010, …)
  - n bits: 2^n combinaciones
- Byte, Kilobyte, Megabyte, Gigabyte, Terabyte, Petabyte (2^{50} bytes), …
  - Zettabyte (1 millón de petabytes)
  - Probabilidad de ganar la Primitiva y caer fulminado por un rayo el mismo día (1 entre 2^{56})
- La codificación también puede ser hexadecimal para reducir fuentes de error y acortar las representaciones de los datos.

$$c: \mathrm{A} \rightarrow \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}^n$$

2.1. Números binarios

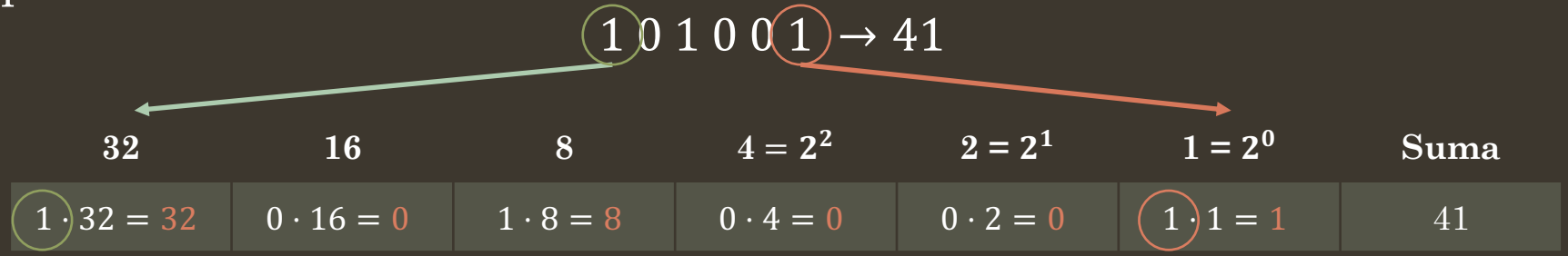
- Ejemplo de codificación:

$$41 \rightarrow 1\,0\,1\,0\,0\,1$$



- Ejemplo de decodificación:

$$1\,0\,1\,0\,0\,1 \rightarrow 41$$



2.1. Números binarios

- Ejemplo de codificación:

$0.671875 \rightarrow 1\ 0\ 1\ 0\ 1\ 1$

$0.671875 \cdot 2 = 1.34375$	$0.34375 \cdot 2 = 0.6875$	$0.6875 \cdot 2 = 1.375$	$0.375 \cdot 2 = 0.75$	$0.75 \cdot 2 = 1.5$	$0.5 \cdot 2 = 1$
$PE = 1$	$PE = 0$	$PE = 1$	$PE = 0$	$PE = 1$	$PE = 1$

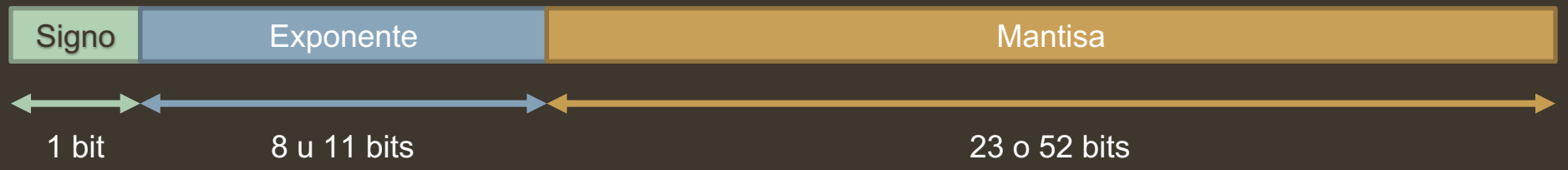
- Ejemplo de decodificación:

$1\ 0\ 1\ 0\ 1\ 1 \rightarrow 0.671875$

$0.5 = 2^{-1}$	$0.25 = 2^{-2}$	0.125	0.0625	0.03125	0.015625	Suma
$1 \cdot 0.5$	$0 \cdot 0.25$	$1 \cdot 0.125$	$0 \cdot 0.0625$	$1 \cdot 0.03125$	$1 \cdot 0.015625$	0.671875

2.1. Números binarios

- A parte de estos existen otras representaciones que siguen estándares particulares como por ejemplo el punto flotante basado en la notación científica con diferentes precisiones (IEEE 754).



$$V_{s/d} = (-1)^S \cdot 2^{(c-127/1023)} \cdot (1+f)$$

c es la representación decimal del exponente y *f* es la representación decimal de la mantisa tomada como decimales.

2.1. Números binarios

$$V_d = (-1)^S \cdot 2^{(c-1023)} \cdot (1 + f)$$

$$c = (1 \cdot 2^{10}) + (1 \cdot 2^{1}) + (1 \cdot 2^{0}) = 1027$$

$$f = (1 \cdot 2^{-1}) + (0 \cdot 2^{-2}) + (1 \cdot 2^{-3}) + (1 \cdot 2^{-4}) + (1 \cdot 2^{-5}) = 0.71875$$

$$V_d = (-1) \cdot 2^4 \cdot (1{,}71875) = -27.5$$

2.2. Lenguajes de bajo nivel

- Las instrucciones de los programas son secuencias de códigos numéricos binarios almacenados en la memoria del computador.
- Se denomina código máquina y es muy complejo para los humanos, por lo cual se requieren lenguajes de programación más cercanos a los programadores.
- El primer acercamiento surge con los lenguajes de bajo nivel o ensamblador. Este lenguaje es dependiente de cada máquina concreta y son instrucciones sencillas e intuitivas para los programadores.

Ensamblador Intel 8086

```
ORG 100h
MOV CX, 5
MOV BX, 1
MOV DL, 2
comienzo:
 MOV AX, BX
 MUL DX
 MOV BX, AX
 LOOP comienzo
 RET
```

```
ORG 100h
mov ax, 10 ;AX=10
mov bx, 00F9h ;BX=0xF9
inc bx ;BX++
add ax, 4 ;AX=AX+4
mov cx,45 ;CX=45
sub cx,cx ;CX=CX-CX
ret
```

2.3. Lenguajes de alto nivel

- Se crean con el propósito de eliminar los problemas de compatibilidad entre maquinas con diversos lenguajes ensambladores.
- Tienen generalmente una curva de aprendizaje menor y están más cercanos al lenguaje natural, lo que facilita la tarea al programador.
- Cuentan con instrucciones más completas que permiten reducir el número de instrucciones a la hora de realizar operaciones complejas. Requieren de compiladores o intérpretes al igual que el de bajo nivel requería el ensamblador.

```
ORG 100h
MOV CX, 5
MOV BX, 1
MOV DL, 2
comienzo:
 MOV AX, BX
 MUL DX
 MOV BX, AX
 LOOP comienzo
 RET
```

cx = 5
bx = 1
dl = 2
while(true):
  ax = bx
  *ax = ax * dx*
  bx = ax

```
 ORG 100h
 mov ax, 10 ;AX=10
 mov bx, 00F9h ;BX=0xF9
 inc bx ;BX++
 add ax, 4 ;AX=AX+4
 mov cx,45 ;CX=45
 sub cx,cx ;CX=CX-CX
ret
```

ax = 10
bx = 249
bx = bx + 1
ax = ax + 4
cx = 45
cx = cx - cx

2.3. Lenguajes de alto nivel

Lenguajes Primitivos

- Fortran
- Cobol
- Basic

Lenguajes Estructurados

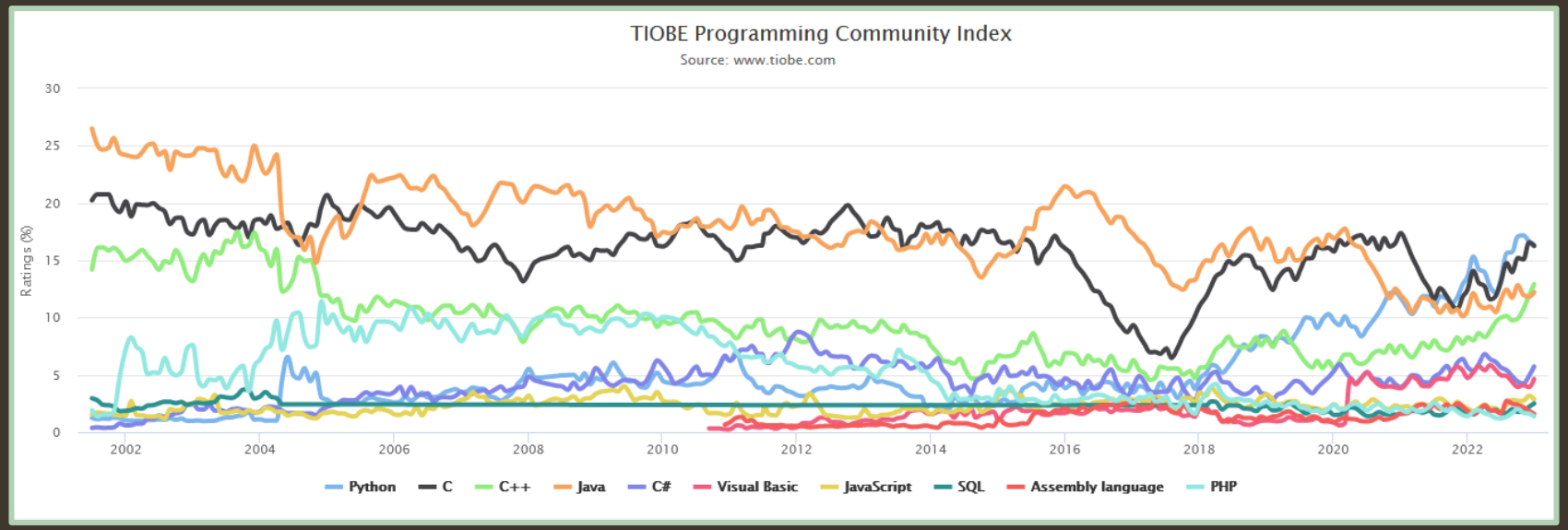
- Pascal
- C
- Ada

Orientados a Objetos

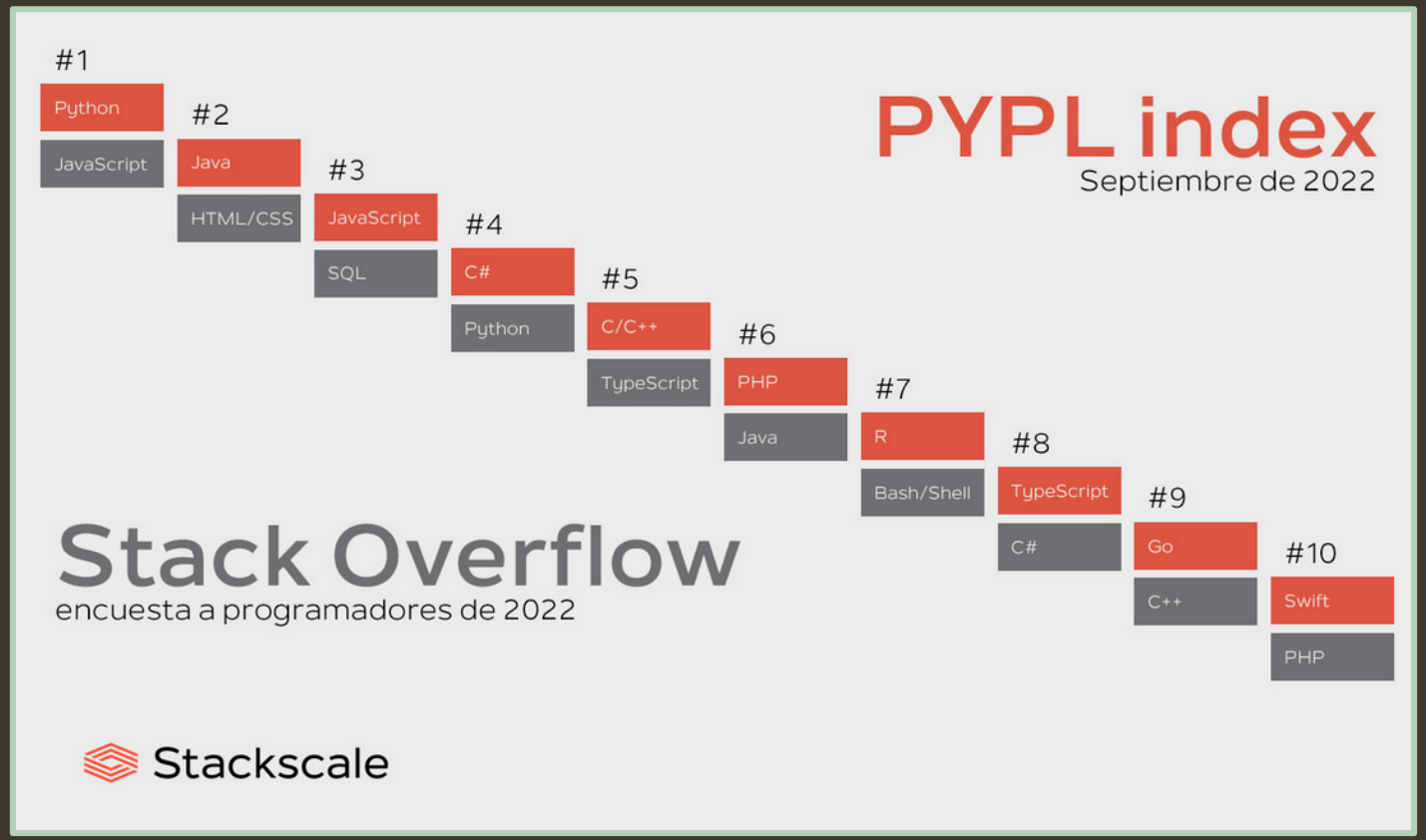
- Java
- C++
- C#

Lenguaje de Scripts

- Python
- R
- JavaScript



2.3. Lenguajes de alto nivel



3. Programación en Python

- Lenguaje de alto nivel de programación interpretado (no requiere compilación pero si un intérprete), dinámico y multiplataforma.
- Soporta parcialmente orientación a objetos y otros paradigmas de programación.
- Interactivo y fácil de interpretar con una curva de aprendizaje menos pronunciada.
- Enfocado a su utilización en: minería de datos, inteligencia artificial, desarrollo web, gráficos 3D y ciencia de datos.
- Tiene detrás una amplia comunidad de desarrolladores y de librerías disponibles que abarcan muchos campos.
- Es de los lenguajes más utilizados y mejor valorados en múltiples encuestas.

3. Programación en Python

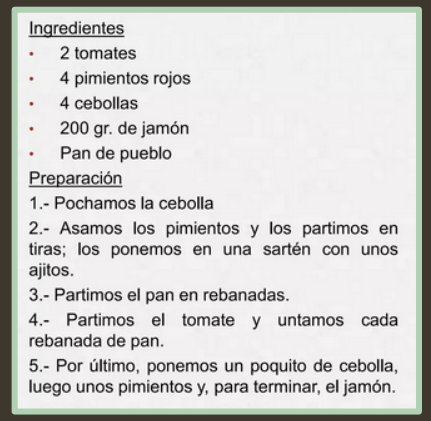
Receta cocina ⟷ Algoritmo

Ingredientes
Utensilios de cocina
Procesos de elaboración
Instrucciones

Literales
Variables y operadores
Funciones y argumentos
Instrucciones

Secuencia de pasos sencillos
Orden concreto
Estructurada para ser comprendida
Concluyen cuando se prepara el plato

Secuencia de instrucciones
Ordenadas con un flujo de control
Alineación y sangría
Finalizan cuando producen el resultado



Ingredientes
- 2 tomates
- 4 pimientos rojos
- 4 cebollas
- 200 gr. de jamón
- Pan de pueblo

Preparación

1.- Pochamos la cebolla

2.- Asamos los pimientos y los partimos en tiras; los ponemos en una sartén con unos ajitos.

3.- Partimos el pan en rebanadas.

4.- Partimos el tomate y untamos cada rebanada de pan.

5.- Por último, ponemos un poquito de cebolla, luego unos pimientos y, para terminar, el jamón.

```
contador = 0
suma = 0
while(contador < 10):
    contador = contador + 1
    if ((contador % 2) == 0):
        suma = suma + contador
        print("Numero sumado")
print(suma)
```

3. Programación en Python

Primitivas	Sintaxis	Semántica
Unidad mínima con significado y valor semántico en un lenguaje.	Parte de la gramática que define las reglas y principios de la combinatoria, definiendo cómo se combinan las palabras.	Estudio de aspectos que dotan de significado, sentido e interpretación a expresiones o representaciones formales.
Hola, comprar, ella, sobre, …	El perro juega con la pelota	Instalé la bomba ayer
=, +, while, if, print, …	producto = 10 * 3	producto = 10 + 3

3. Programación en Python

Primitivas	Sintaxis	Semántica
Unidad mínima con significado y valor semántico en un lenguaje.	Parte de la gramática que define las reglas y principios de la combinatoria, definiendo cómo se combinan las palabras.	Estudio de aspectos que dotan de significado, sentido e interpretación a expresiones o representaciones formales.

Errores comunes		
Desconocimiento	Estructuras incorrectas	Comportamientos no esperados
Intento de sobreescritura	Instrucciones incompletas	El programa finaliza inesperadamente
Utilización incorrecta	Paréntesis, sangrías y corchetes	El programa nunca finaliza
	Errores de ejecución	Resultados anómalos

3.1. Clases de objetos y variables

Objeto: *instancia o solicitud de una clase que puede ser* ***escalar*** *(divisible) o* ***no escalar****.*

Tipo	Notación	Ejemplo
Entero	int	1234
Punto Flotante	float	12.34
Complejo	complex	1+2j
Booleano	bool	True/False
Caracter	chr	'A'

Tipos Especiales: -inf, inf, None, NaN

```
[1]: type(1234)
[1]: int

[2]: type(12.34)
[2]: float

[3]: type(1+2j)
[3]: complex

[4]: type(True)
[4]: bool

[5]: type('a')
[5]: str

[10]: print(chr(65))
      A
```

3.1. Clases de objetos y variables

Objeto: *instancia o solicitud de una clase que puede ser* ***escalar*** *(divisible) o* ***no escalar****.*

- Entero:
  - Precisión ilimitada.
  - Aritmética exacta.
- Punto flotante:
  - Precisión de unos 15 dígitos y rango ±1.7E+308.
  - Ocupa 64 bits.
  - Aritmética aproximada con posibles errores de redondeo (1-(5/3-4/3)*3).
- Complejos:
  - Requieren más coste computacional.

3.1. Clases de objetos y variables

Casting: *conversión de la clase de un objeto a otra diferente.*

- Notación:

Tipo(objeto)

- Permite cambiar la clase de un objeto para realizar operaciones específicas (indexar, operar, representar, etc.).
- Hay que tener cuidado ya que puede conllevar problemas entre clases (ej. str e int).

```
[24]: print(type(65))
      65

      <class 'int'>
[24]: 65

[22]: print(type(float(65)))
      float(65)

      <class 'float'>
[22]: 65.0

[21]: print(type(complex(65)))
      complex(65)

      <class 'complex'>
[21]: (65+0j)

[20]: print(type(str(65)))
      str(65)

      <class 'str'>
[20]: '65'
```

3.1. Clases de objetos y variables

Variable: *asignaciones de memoria que contienen objetos.*

- Son útiles a la hora de nombrar valores.
- Permiten la reutilización de un valor concreto.
- Creadas cuando su valor es asignado por primera vez:

Variable = objeto

- Su valor puede variar durante la ejecución del código:

Variable = nuevo_objeto

- No pueden usarse nombres de variable reservados.
- Pueden ser locales o globales.

```
[5]: temperatura = 25.0
     print(temperatura)
     type(temperatura)

     25.0
[5]: float

[4]: temperatura = 30
     print(temperatura)
     type(temperatura)

     30
[4]: int

[8]: for = 10

       Cell In [8], line 1
         for = 10
             ^
     SyntaxError: invalid syntax
```

3.2. Operadores matemáticos y lógicos

Operador: *símbolo modificador o comparador utilizado entre dos objetos.*

Operador	Operación	Ejemplo	Resultado	Prioridad
**	Exponenciación	2**3	8	1
*	Multiplicación	3.5*2	7.0	2
/	División	5/2	2.5	2
//	Cociente	7//2	3	2
%	Resto / Módulo	7%2	1	2
+	Suma	4+7	11	3
-	Resta	5-3j-19	-14-3j	3

Operadores Matemáticos se ejecutan de izquierda a derecha excepto la exponenciación.

3.2. Operadores matemáticos y lógicos

Contracción Operador-Asignación: *símbolo modificador de una variable que combina una operación aritmética y una de asignación entre dos objetos.*

Operador	Operación	Ejemplo	Resultado
**=	Exponenciación	x**=3	8
*=	Multiplicación	x*=2	4
/=	División	x/=2	1
//=	Cociente	x//=2	1
%=	Resto / Módulo	x%=2	0
+=	Suma	x+=7	9
-=	Resta	x-=19	-17

En el ejemplo de la tabla se ha utilizado un valor de x = 2

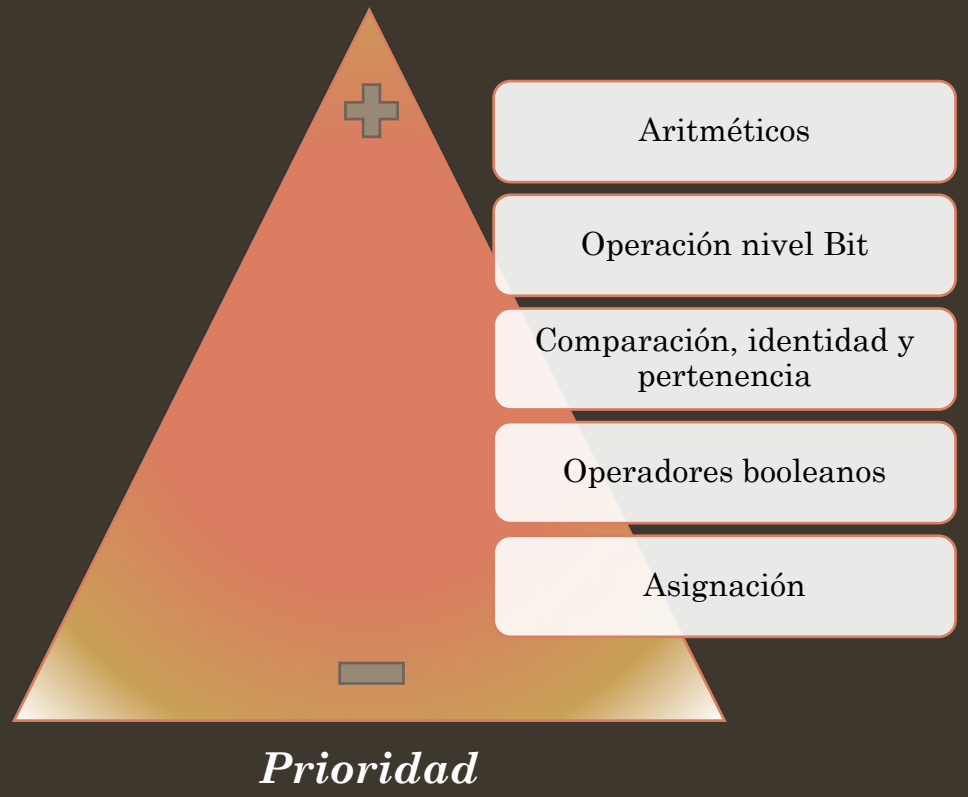
3.2. Operadores matemáticos y lógicos

Operador: *símbolo modificador o comparador utilizado entre dos objetos.*

Operador	Operación	Ejemplo	Resultado	Prioridad
>, >=	Mayor (o igual)	3 > 2	True	1
<, <=	Menor (o igual)	3 < 2	False	1
==	Igual	'hola' == 'hola'	True	1
&, and	And	True & False	False	1
|, or	Or	True | False	True	1
!=	Distinto	3.0 != 3	False	1
not	Not	not 1	False	1

Operadores Lógicos se ejecutan de izquierda a derecha tomando True valor 1 y False 0.

3.2. Operadores matemáticos y lógicos



+ - * / % // **

| & ^ << >> -

is in > < == !=

and or not

= += -= *= %=

3.3. E/S Teclado

Entrada: *lectura de datos provenientes del usuario.*

Salida: *presentación al usuario de datos o resultados.*

almacenamiento_entrada = input("mensaje")

print(variable_mostrada)

```
nombre = input("Dime tu nombre: ")
apellido1 = input("Dime tu primer apellido: ")
apellido2 = input("Dime tu segundo apellido: ")
print("Encantado de conocerte, "+str(nombre)+" "+str(apellido1)+" "+str(apellido2))
```

```
Dime tu nombre:  Javier
Dime tu primer apellido:
```

```
nombre = input("Dime tu nombre: ")
apellido1 = input("Dime tu primer apellido: ")
apellido2 = input("Dime tu segundo apellido: ")
print("Encantado de conocerte, "+str(nombre)+" "+str(apellido1)+" "+str(apellido2))
```

```
Dime tu nombre:  Javier
Dime tu primer apellido:  González
Dime tu segundo apellido:  Villa
Encantado de conocerte, Javier González Villa
```