PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

INTRODUCCIÓN A LA PROGRAMACIÓN

Funcionamiento del computador

- Resumen cátedra pasada
  - Algoritmos
  - Diagramas y pseudocódigos
  - Algoritmos para números binarios

- Contenidos de esta cátedra
  - Introducción a Python
  - Variables y valores
  - Entrada y salida
  - Introducción a expresiones

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

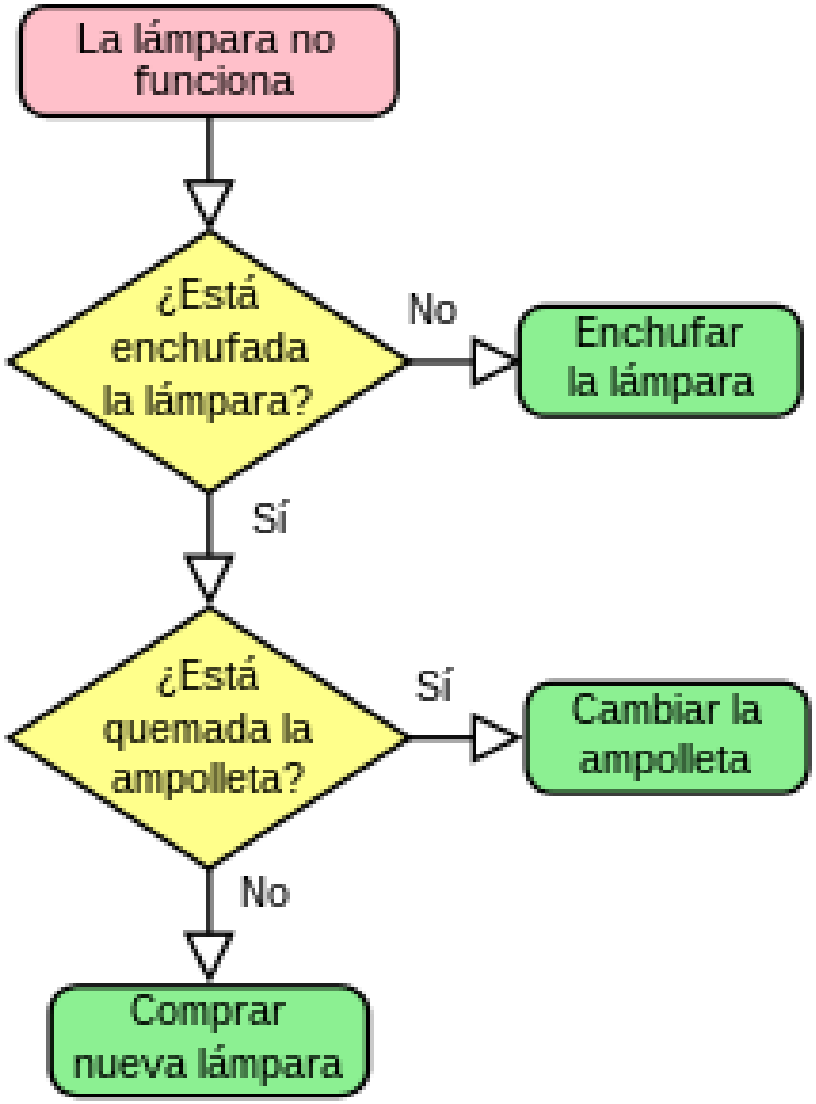
¿Qué es un algoritmo?

- Un conjunto de reglas predefinidas que, dado una *entrada* o *insumos* específicos, obtiene una *salida* o *resultado* esperado/específico
- Conjunto de pasos que toman un *input* y resultan en un *output*
- *RAE: Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema*
- Procedimiento

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Diagramas de flujo

- Muy prácticos para explicar procesos
- Diversas notaciones
  - Formas de representar (escribir, dibujar) las ideas
- ¿Qué hace el diagrama de la izquierda?
- Podemos usar diagramas de flujo para guiar nuestra programación



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

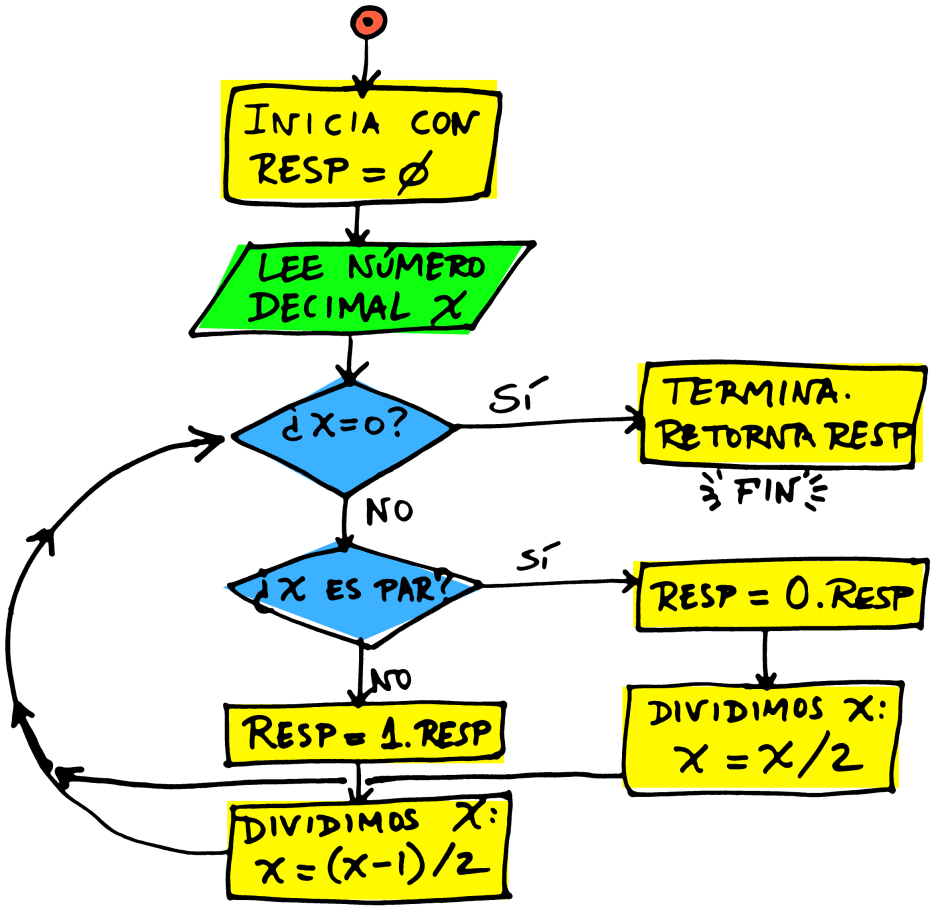
Pseudocódigo

```
Procedimiento Ordenar (L)
    //Comentario: L = (L_1, L_2, ..., L_n) es una lista con n elementos//
    k ← 0;
    Repetir
        intercambio ← Falso;
        k ← k + 1;
        Para i ← 1 Hasta n − k Con Paso 1 Hacer
            Si L_i > L_{i+1} Entonces
                intercambiar (L_i, L_{i+1})
                intercambio ← Verdadero;
            Fin Si
        Fin Para
    Hasta Que intercambio = Falso;
Fin Procedimiento
```

- A veces conviene escribir los algoritmos como **diagramas de flujo** y/o como **pseudocódigo**
- El pseudocódigo es como si programáramos, pero el lenguaje usado es **informal**
- Sirve para comunicar la idea tras el algoritmo
- Se puede traducir a **código**; en nuestro caso, esto sería **Python 3**

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Conversión de decimal a binario



El diagrama de flujo describe el proceso para obtener la representación binaria (base 2) de un número decimal (base 10).

Dividimos por 2 repetidamente (división entera). Si nuestro número es impar, colocamos 1 en la respuesta. De lo contrario, colocamos 0. Repetimos hasta llegar a cero.

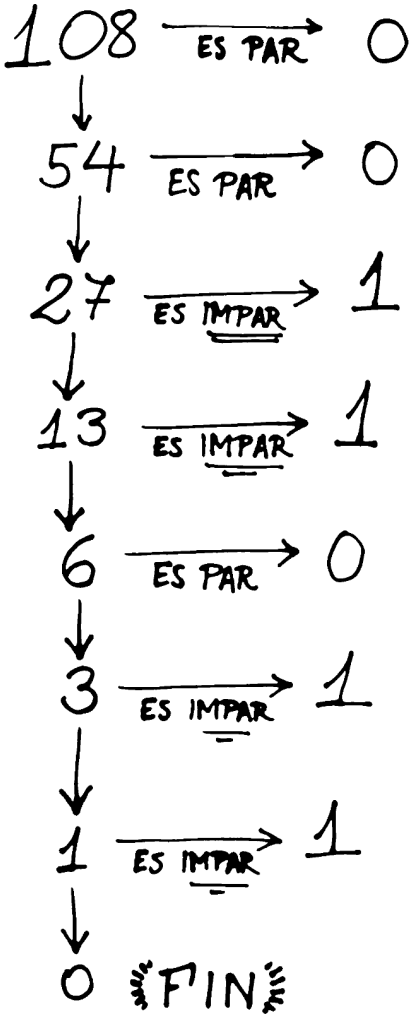
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Convirtiendo a binario

Respuesta

1101100

- 1 se divide por 2 (división entera), lo que resulta en 0.
- Como se ha llegado al cero, el algoritmo se detiene.
- El resultado es 110100.



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Conversión de binario a decimal

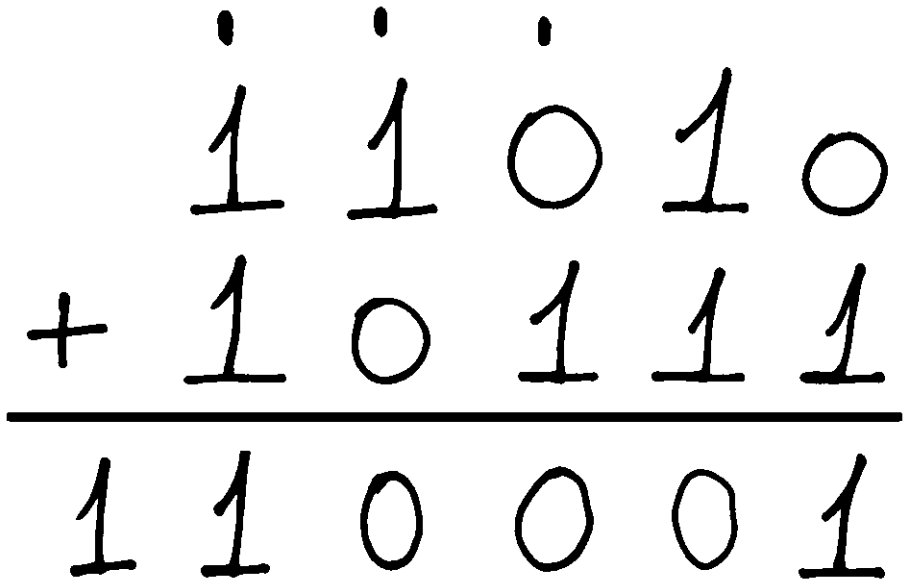
BINARIO	1 1 0 1 1 0 0
POTENCIAS DE 2	64 32 16 8 4 2 1
SUMA	64+32+0+8+4+0+0
RESULTADO	108

- El resultado es 108, tal como el número inicial del caso anterior (convertir decimal a binario)
- Se verifica la integridad del algoritmo

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Suma de números binarios

- El número binario 11010 es 26 en decimal
- El número binario 10111 es 23 en decimal
- Su resultado, 110001, es 49 en decimal
- Efectivamente 26 + 23 = 49



Introducción a Python

Preámbulo

- **Código fuente**: cuando programamos, escribimos los programas en **archivos de texto plano**, los que serán interpretados o compilados luego
- Para medir el *tamaño* de los programas, se suele considerar el número de carácteres, el número de instrucciones o el número de líneas
- **Compilador**: programa que toma el código fuente y genera un programa ejecutable
- **Intérprete**: programa que inmediatamente ejecuta el código fuente; a veces permite ejecutar línea por línea

Niveles de lenguaje

- Se dice que los lenguajes tienen *niveles* de acuerdo a su cercanía a la *máquina* -- ¡es un *espectro continuo*!
- **Bajo nivel**: los lenguajes de bajo nivel son aquellos cercanos a la máquina; **muchas instrucciones hacen poco**
  - Lenguaje de ensamblador: es el nivel más bajo, depende de la CPU
  - Lenguaje C: muy bajo nivel, se solicita y libera memoria como parte de las instrucciones
- **Alto nivel**: los lenguajes de alto nivel son aquellos distantes a la máquina; **pocas instrucciones hacen mucho**
  - Python: muy alto nivel, permite ignorar
  - BASIC, Pascal, Scala, etc

Paradigmas de programación

- **Imperativo**: se ejecuta línea a línea, se permite saltar entre líneas
- **Estructurado**: es el paradigma que **veremos en este curso**; tiene bien ordenado el flujo del programa (saltos condicionales, repeticiones, subrutinas)
- **Orientado a objetos**: los valores son *objetos*, que se pueden programar para realizar acciones específicas
- **Funcional**: está basado en llamadas a funciones (subrutinas); aquí la *recursividad* pasa a ser esencial
- También hay paradigmas especializados para tareas específicas (ej. bases de datos, música, ecuaciones, lógica)

Python

- Alto nivel
- Interpretado
  - También puede ser compilado
- Multi-paradigma:
  - Estructurado (este curso)
  - Orientado a objetos
  - Funcional
- Muy popular
- Muchas formas de programarlo

Usando online-python.com

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

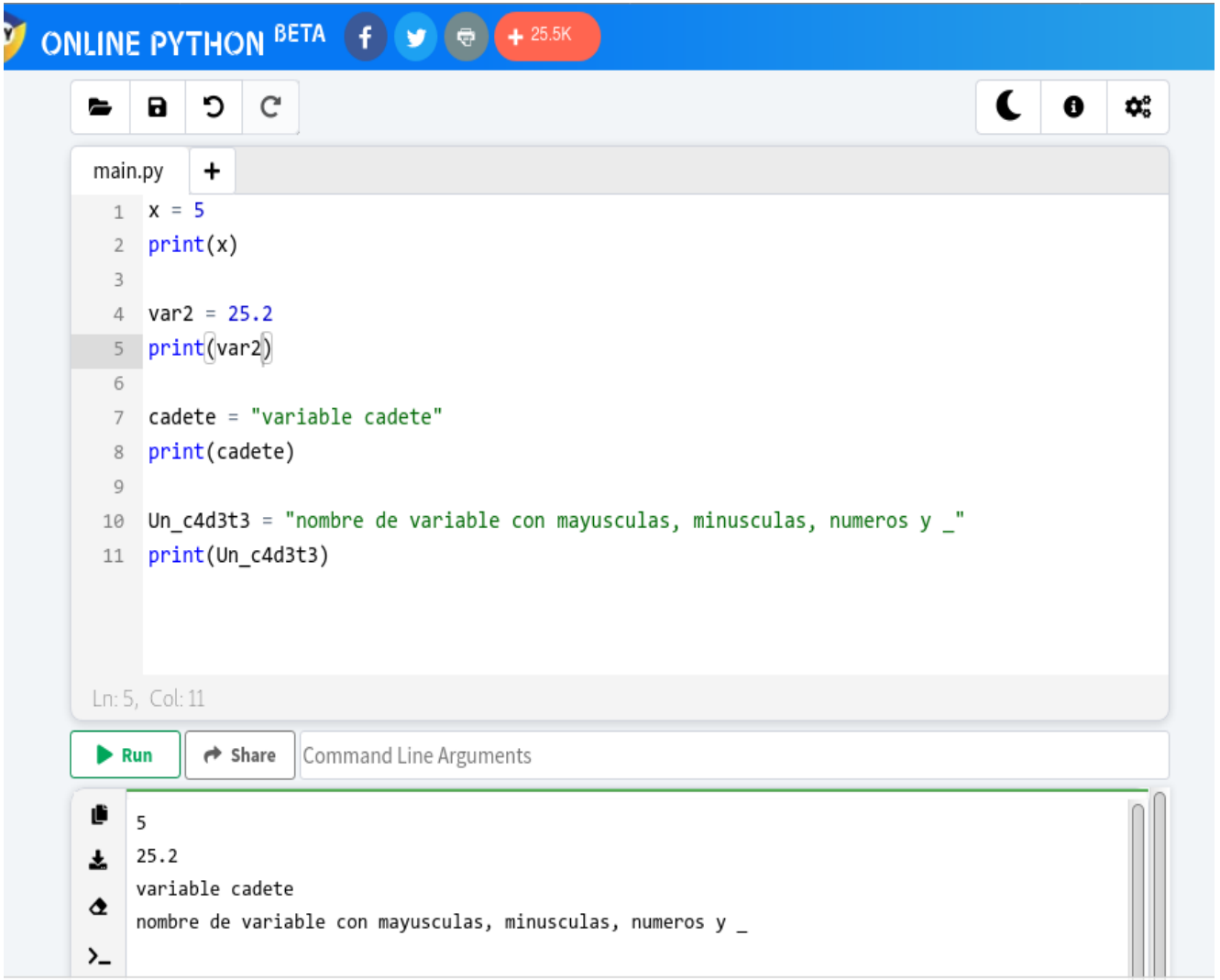
Ejemplos sencillos

Algunos ejemplos con *Online Python*

Los nombres de variable pueden ser escritos de manera muy flexible.

El objetivo es recordar después dónde hemos memorizado algún dato relevante para el programa.

Nota: al escribir `variable = algo` la estamos definiendo y le *asignamos* un valor



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Observación

Si dos variables tienen nombres parecidos (ej. `variable` y `Variable`), para Python son variables completamente distintas.

Python no se confunde.

Y si *apelamos* a una variable con un nombre que no existe aún (no la hemos definido), entonces Python lanzará un mensaje de error.

main.py

```
1  # definimos dos variables que se llaman parecido, pero son distintas
2
3  variable = 100
4  Variable = 200
5
6  # Python no se confunde y reconoce las variables;
7  #  imprimira 100 en una linea y luego imprimira 200 en la segunda
8
9  print(variable)
10 print(Variable)
```

n: 7, Col: 66

Run | Share | Command Line Arguments

```
100
200
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Operadores de asignación

OPERADOR	DESCRIPCIÓN
=	a = 5. El valor 5 es asignado a la variable a
+=	a += 5 es equivalente a a = a + 5
-=	a -= 5 es equivalente a a = a - 5
*=	a *= 3 es equivalente a a = a * 3
/=	a /= 3 es equivalente a a = a / 3
%=	a %= 3 es equivalente a a = a % 3
**=	a **= 3 es equivalente a a = a ** 3

main.py +

```
1  # definimos una variable con un valor cualquiera
2  x = 100
3  print(x)   # mostramos su valor
4
5  # actualizamos usando +=
6  x += 50    # equivale a escribir x = x + 50, que es x = 100 + 50
7  print(x)   # mostramos su nuevo valor: 150 ahora
8
9  # actualizamos usando /=
10 x /= 25    # equivale a escribir x = x / 25, que es x = 150 / 25
11 print(x)   # mostramos su nuevo valor: 6.0
```

Ln: 11, Col: 43

Run | Share | Command Line Arguments

```
100
150
6.0
```

Ejemplo de asignación

El código a la derecha demuestra varios tipos de asignación:

`=` (simple)
`+=` (suma un valor)
`/=` (divide por un valor)

El código de la derecha equivale a escribir

```
x = 100
print(x)
x = x + 50
print(x)
x = x / 50
print(x)
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Operadores aritméticos

OPERADOR	DESCRIPCIÓN	USO
+	Realiza Adición entre los operandos	12 + 3 = 15
-	Realiza Substracción entre los operandos	12 - 3 = 9
*	Realiza Multiplicación entre los operandos	12 * 3 = 36
/	Realiza División entre los operandos	12 / 3 = 4
%	Realiza un módulo entre los operandos	16 % 3 = 1
**	Realiza la potencia de los operandos	12 ** 3 = 1728
//	Realiza la división con resultado de número entero	18 // 5 = 3

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Ejemplo de operaciones

Las fórmulas del código que se ve a la derecha terminan resultando en valores diferentes (ver recuadro de abajo).

¿Por qué ocurre esto? Por las prioridades de las operaciones y por los paréntesis.

Mayor prioridad: **
Alta prioridad: *, /
Media prioridad: +, -
Más baja prioridad: =

main.py +

```
1  # cada formula a continuacion se evalua de forma diferente
2  # jugando entre prioridades y parentesis -- por que?
3
4  x = 1 + 5 / 4 + 2 * 3 - 1
5  print(x)   # mostramos su valor
6
7  x = (1 + 5) / (4 + 2 * 3 - 1)
8  print(x)   # mostramos su valor
9
10 x = (1 + 5) / 4 + 2 * (3 - 1)
11 print(x)   # mostramos su valor
12
13 x = (1 + 5) / (4 + 2) * (3 - 1)
14 print(x)   # mostramos su valor
```

Ln: 13, Col: 22

Run | Share | Command Line Arguments

```
7.25
0.6666666666666666
5.5
2.0
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Operadores relacionales o de comparación

OPERADOR	DESCRIPCIÓN	USO
>	Devuelve True si el operador de la izquierda es mayor que el operador de la derecha	12 > 3 devuelve True
<	True si el operador de la derecha es mayor que el operador de la izquierda	12 < 3 devuelve False
==	True si ambos operandos son iguales	12 == 3 devuelve False
>=	True si el operador de la izquierda es mayor o igual que el operador de la derecha	12 >= 3 devuelve True
<=	True si el operador de la derecha es mayor o igual que el operador de la izquierda	12 <= 3 devuelve False
!=	True si ambos operandos no son iguales	

main.py

```
1  a = 5
2  b = 10
3
4  # vimos las operaciones and, or y not en clases
5  z = ( a > b ) and ( a != b )     # False and True --> False
6  print( z )
7
8  # ahora probamos con or
9  z = ( a > b ) or ( a != b )     # False or True --> True
10 print( z )
11
12 # not mayo
13 mayo = ( a < b ) or ( b < a )   # False or False --> False
14 print( not mayo )
```

Ln: 14, Col: 19

Run | Share | Command Line Arguments

```
False
True
False
```

Los operadores de comparación (<, <=, >, >=, == y !=) entregan valores de *verdad*, los cuales son **True** y **False**.

True significa verdadero. False significa falso.

En clases vimos, además, cómo combinar los valores de *verdad* (booleanos). Hablamos de **and**, **or** y **not**.

El ejemplo de la lámina muestra qué hacen.

Cierre

Preguntas

- ¿Qué es una variable? ¿Para qué sirve?
- ¿Qué es un valor?
- ¿Qué es una expresión?
- ¿Qué imprimen los siguientes *print*?

```
print( 1000 * 5 / 10 )
print( 'hola', "mundo" )
print( input() )
```