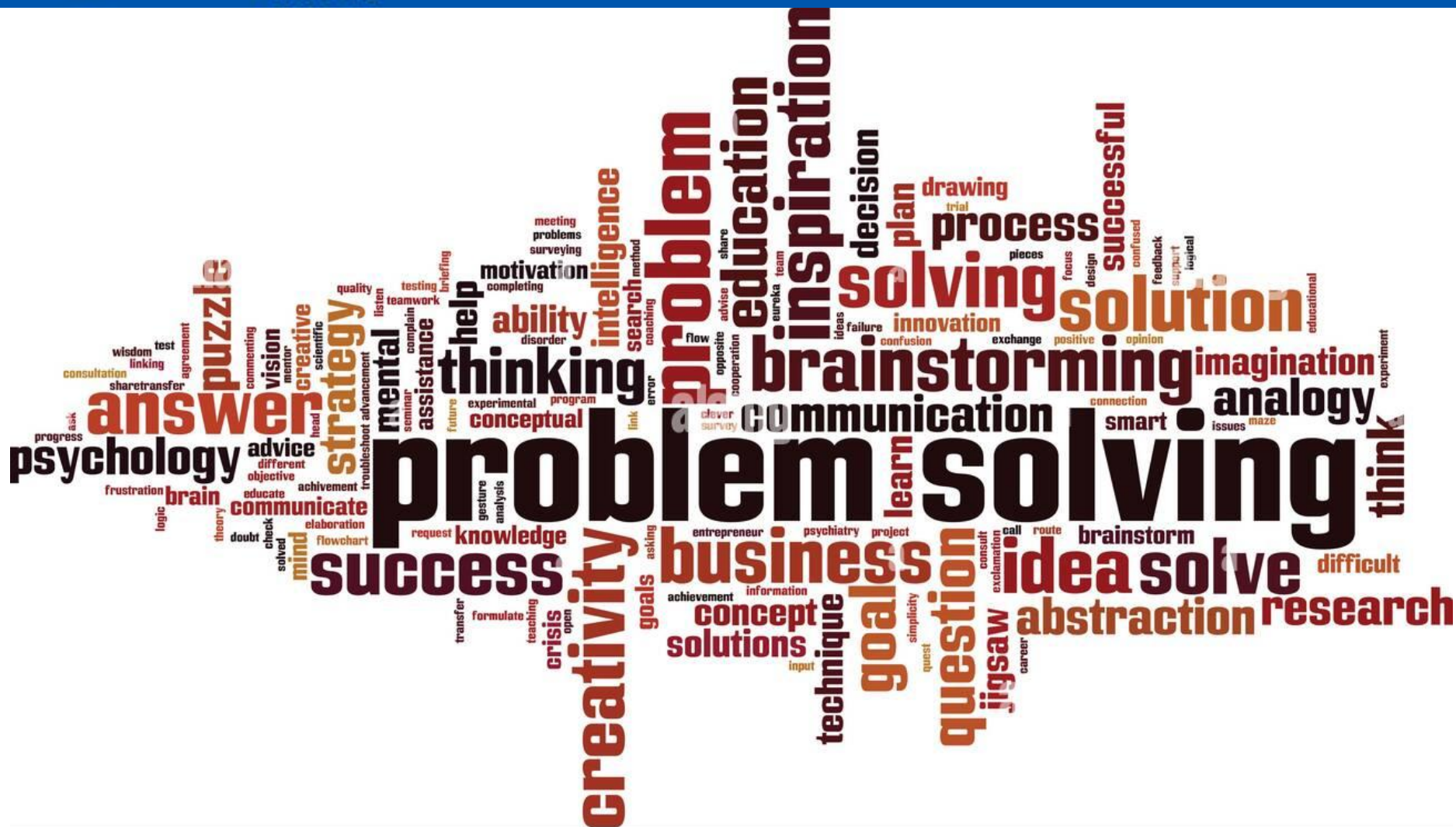


Fundamentos de Programación

Prof. Dr. Roberto Muñoz S. (Sec 1, Lab 502)
Prof. Dr. Rodrigo Olivares O. (Sec 2, Lab 401)
Prof. Víctor Ríos (Sec 3, Lab 303)
Prof. Pablo Olivares (Sec 4, Lab 301)

Escuela de Ingeniería en Informática
Facultad de Ingeniería
Universidad de Valparaíso

Resolución de problemas



Es un proceso que incluye **creatividad** y métodos de **ingeniería**.



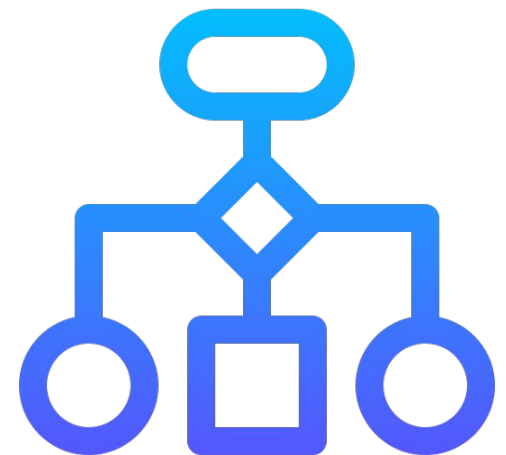
Hoy

- Definición de Algoritmo
- Constantes, Variables, Expresiones y Sentencias
- Expresiones
- Reglas de precedencia

Algoritmo

Secuencia **ordenada** de pasos, exentos de **ambigüedad** tal que, al llevarse a cabo con **fidelidad**, dará como **resultado** que se realice la tarea para la que se ha diseñado en un **tiempo finitos**.

Algoritmo



Constantes, Variables, Expresiones y Sentencias

- Valores fijos, como números, letra, cadenas, son llamadas **constantes**.
- Una palabra es una cadena de caracteres o **string**.
- Las cadenas constantes usan comillas sencillas (') o dobles (")

- Una variable es un lugar con “**nombre**” en la memoria donde un programador puede almacenar datos.
- Para recuperar el dato (valor) se utiliza el “**nombre**” de la variable.

- Los programadores pueden escoger los nombres de las variables.
- Puede cambiar el contenido de una variable con otra sentencia.

```
x = 12.2  
y = 14  
x = 100 * x
```

x	12.2	100
y	14	

100 = x

Reglas de Python para nombres de Variables

- Deben empezar con una letra o *guión bajo* _
- Deben consistir en letras y números y guiones bajos
- Diferencian entre mayúsculas y minúsculas
- **Buenos:** spam eggs spam23 _speed
- **Incorrectos:** 23spam #sign var.12
- **Diferentes:** spam Spam SPAM

No pueden usar algunas **palabras reservadas**
como nombres de variables / identificadores:

and del for is raise assert elif
from lambda return break else
global not try class except if or
while continue exec import pass
yield def finally in ~~print~~ as with

Asignaciones (Python)

`x = 2` ← Enunciado de **asignación**

`x = x + 2` ← **Asignación** con **expresión**

`print(x)` ← Sentencia de **impresión**

variable

operadores

palabra reservada

constante

- Asignamos un valor a una variable usando el operador de **asignación** (=).
- Una **asignación** consiste en una **expresión del lado derecho** y una **variable** que almacene el resultado.

$$x = 3.9 * x * (1 - x)$$

x = 0.6

x = 3.9 * **x** * (1 - **x**)

0.936

El lado derecho es una expresión. Una vez que la expresión se evalúa, el resultado se almacena en (el lugar asignado a) **x**.

Expresiones

- **Aritméticas** para la construcciones de expresiones numéricas. **Regresa un valor resultante de la evaluación de la expresión.**
- **Relacionales** describe la relación entre **dos componentes**. Regresa un valor **booleano** (**verdadero** o **falso**).
- **Lógicas** para el desarrollo de proposiciones lógicas complejas.

- El símbolo **asterisco** es **multiplicación**.
- Potencia se escribe diferente que en matemáticas.
- ¿**Módulo**? Es el **resto** de una **división entera**.

Operador	Operación
+	Adición
-	Sustracción
*	Multiplicación
/	División
**	Potencia
%	Resto

%



Obs: **NO** representa porcentajes.

- El menor o igual que y el mayor o igual que, **incluyen** una evaluación de **expresión lógica**.

Operador	Operación
$<$	Menor que
$<=$	Menor o igual que
$>$	Mayor que
$>=$	Mayor o igual que
$==$	Igual
$!=$	Distinto

- Permiten desarrollar proposiciones complejas.

Operador	Operación
and	Conjunción (Y)
or	Disyunción (O)
not	Negación (NO)

Reglas de precedencia

Reglas de precedencia (expresiones aritméticas)

- Orden descendiente:
 - Los paréntesis siempre son respetados
 - Potencias
 - Multiplicación, División, y Restante
 - Adición y Sustracción
 - De izquierda a derecha
- Paréntesis
- Potencia
- Multiplicación/División
- Adición/Sustracción
- Izquierda a derecha

Reglas de precedencia (expresiones relaciones)

- Orden indiferente:
 - Izquierda a derecha

Reglas de precedencia (expresiones lógicas)

- Orden descendente:
 - NO (**not**)
 - Y (**and**)
 - O (**or**)

Ejemplo, resolver para n = 2

$15 + 59 * 75 / 9 < 2 ** 3 ** 2 \vee (15 + 59) * 75 \% N == 1$

Ejemplo, respuesta para n = 2

Reglas de precedencia

15 + 59 * 75 / 9 < 2 ** 3 ** 2 Y (15 + 59) * 75 % N == 1
15 + 59 * 75 / 9 < 2 ** 3 ** 2 Y 74 * 75 % N == 1
15 + 59 * 75 / 9 < 2 ** 9 Y 74 * 75 % N == 1
15 + 59 * 75 / 9 < 512 Y 74 * 75 % N == 1
15 + 4425 / 9 < 512 Y 74 * 75 % N == 1
15 + 491 < 512 Y 74 * 75 % N == 1
15 + 491 < 512 Y 5550 % N == 1
15 + 491 < 512 Y 5550 % 2 == 1
15 + 491 < 512 Y 0 == 1
506 < 512 Y 0 == 1
Verdadero Y 0 == 1
Verdadero Y Falso
Falso

$$x = 1 + 2 * 3 - 4 / 5$$

- Recuerde las reglas de arriba abajo.
- Al escribir código - **use paréntesis para definir la precedencia** de una expresión.
- Al escribir código - mantenga las expresiones lo suficientemente **simples** de manera que sean fáciles de entender.
- **Separe** operaciones matemáticas largas para hacerlas más claras.

Ejercicio - Validar RUN - DV

El RUN (Rol Único Nacional) es un elemento identificador de personas en Chile y se usa desde 1969. Consiste en una combinación entre un número y un dígito verificador, que varía entre el 0 al 9 o la letra K.

El RUN es **único** e **irrepetible**, y su dígito verificador se calcula a partir del mismo número. El dígito verificador existe para evitar engaños y suplantaciones de identidad, y es calculado con un **algoritmo**, con una secuencia de instrucciones de cálculos aritméticos (expresiones).

Para validar el RUN, realice los siguientes pasos (ejemplo: RUN = **15.742.808**, DV = **K**):

- 1. Multiplicar cada dígito del rut, de atrás hacia adelante, por 2, 3, 4, ..., 7, 2, 3**
($8 * 2$), ($0 * 3$), ($8 * 4$), ($2 * 5$), ($4 * 6$), ($7 * 7$), ($5 * 2$), ($1 * 3$)
- 2. Como segundo paso debe sumar todas las multiplicaciones parciales.**
 $16 + 0 + 32 + 10 + 24 + 49 + 10 + 3$
- 3. De esta suma, debe sacar el resto de la división por 11.**
 $144 \% 11$, el resultado 1
- 4. Por último, el dígito verificador es el resultado de la resta entre 11 y el resultado del paso anterior.**
 $11 - 1$, **SI** el resultado es 10 **ENTONCES** como el resultado NO ES un dígito, se cambia por K.
 $11 - 0$, **SI** el resultado es 11 **ENTONCES** como el resultado NO ES un dígito, se cambia por 0.

```
print("Ingrese la decena de millón de su RUN: ")  
decMo = int(input())
```

```
print("Ingrese la unidad de millón de su RUN: ")  
uniMo = int(input())
```

```
print("Ingrese la centena de mil de su RUN: ")  
cenMi = int(input())
```

```
print("Ingrese la decena de mil de su RUN: ")  
decMi = int(input())
```

```
print("Ingrese la unidad de mil de su RUN: ")  
uniMi = int(input())
```

```
print("Ingrese la centena de su RUN: ")  
cen = int(input())
```

```
print("Ingrese la decena de su RUN: ")  
dec = int(input())
```

```
print("Ingrese la unidad de su RUN: ")  
uni = int(input())
```

Código en:

bit.ly/colab116



```
uni = uni * 2
dec = dec * 3
cen = cen * 4
uniMi = uniMi * 5
decMi = decMi * 6
cenMi = cenMi * 7
uniMo = uniMo * 2
decMo = decMo * 3
```

```
suma = uni + dec + cen + uniMi + decMi + cenMi + uniMo + decMo
```

```
resto = suma % 11
```

```
dv = 11 - resto
```

```
if dv == 11:
    print("DV: 0")
else:
    if dv == 10:
        print("DV: K")
    else:
        print("DV: ", dv)
```

Código en:

bit.ly/colab116



Tipos de datos

Tipo de dato

Un tipo de dato, es un atributo de los datos que indica al computador (o al programador) sobre la clase de datos que se va a manejar.

Esto incluye imponer restricciones en los datos, como qué valores pueden tomar y qué operaciones se pueden realizar.

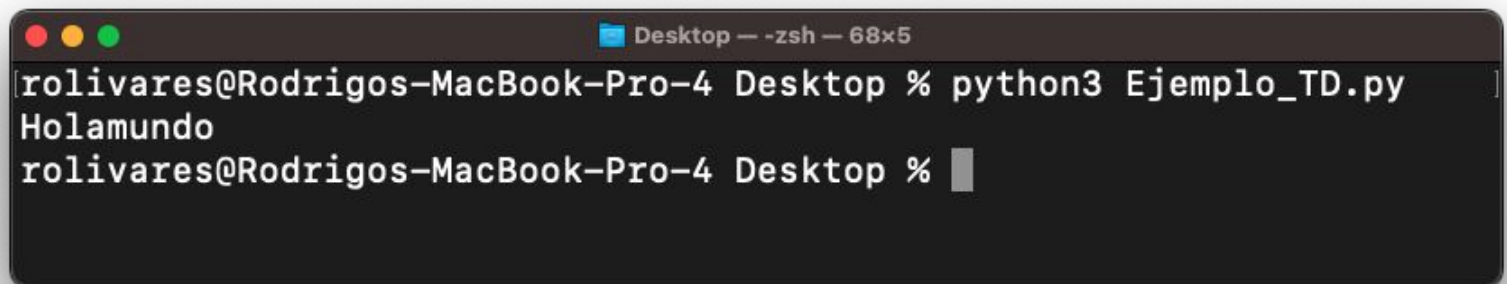
Más simple...

*Son un conjunto (o dominio) de valores
válidos junto con sus operaciones asociadas.*

La importancia del tipo

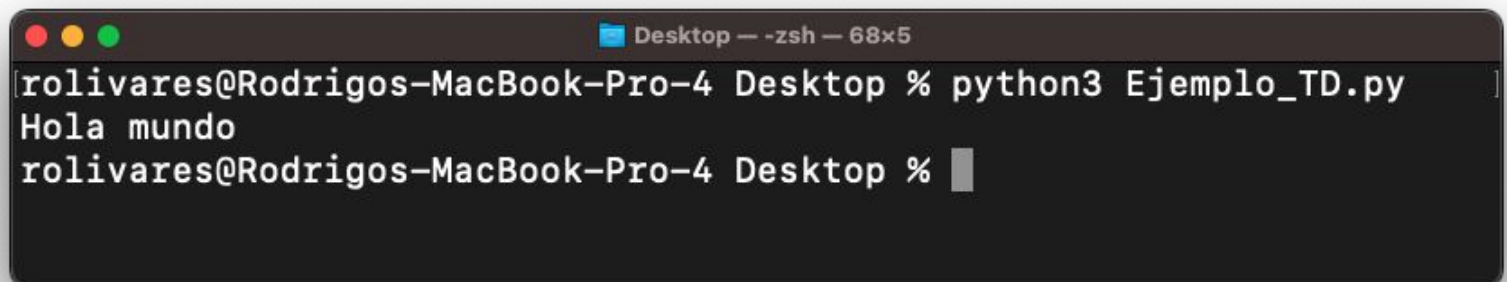
- Python define tipos de datos para cada elemento.
- Existen algunas operaciones prohibidas entre variables de diferentes tipos.
 - p.e., no es posible “añadir 1” a una cadena.
- Podemos preguntar a Python el tipo del elemento, usando la función **type()**.

```
print("Hola" + "mundo")
```



```
Desktop — zsh — 68x5
rolivares@Rodrigos-MacBook-Pro-4 Desktop % python3 Ejemplo_TD.py
Holamundo
rolivares@Rodrigos-MacBook-Pro-4 Desktop %
```

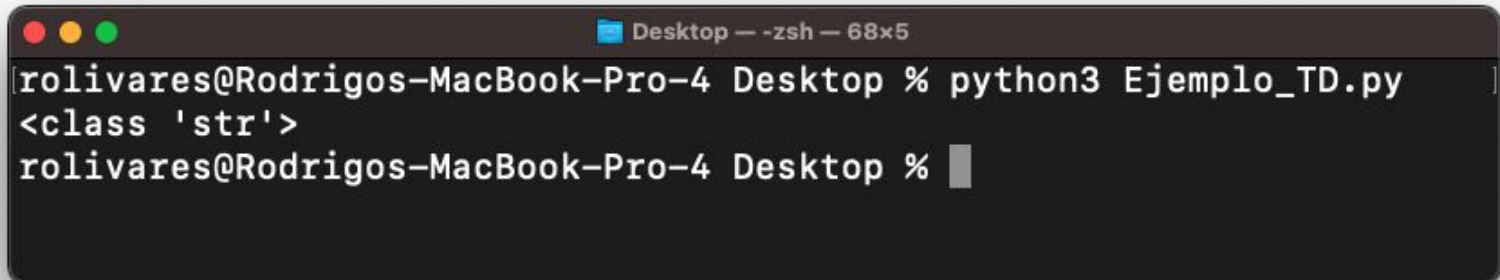
```
print("Hola" + " mundo")  print("Hola " + "mundo")
```



```
Desktop — zsh — 68x5
rolivares@Rodrigos-MacBook-Pro-4 Desktop % python3 Ejemplo_TD.py
Hola mundo
rolivares@Rodrigos-MacBook-Pro-4 Desktop %
```

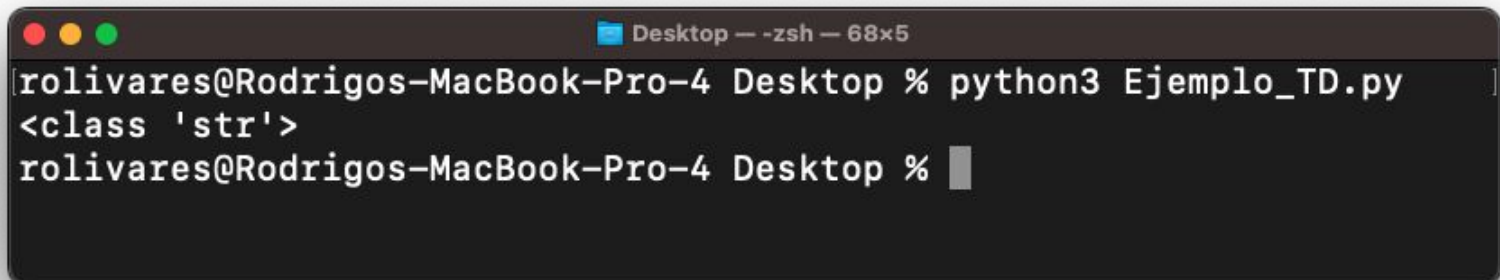
¿Por qué puedo “sumar” letras?

```
print(type("Hola " + "mundo"))
```



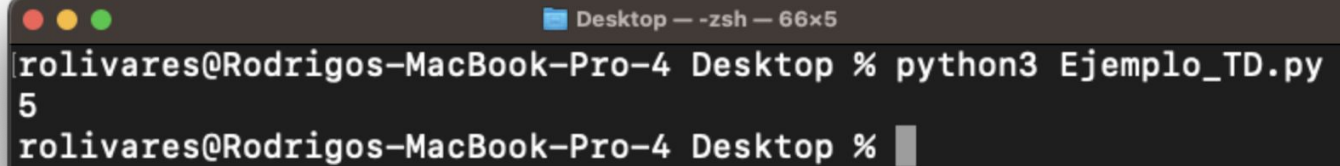
```
Desktop — zsh — 68x5
rolivares@Rodrigos-MacBook-Pro-4 Desktop % python3 Ejemplo_TD.py
<class 'str'>
rolivares@Rodrigos-MacBook-Pro-4 Desktop %
```

```
saludo = "Hola " + "mundo"
print(type(saludo))
```



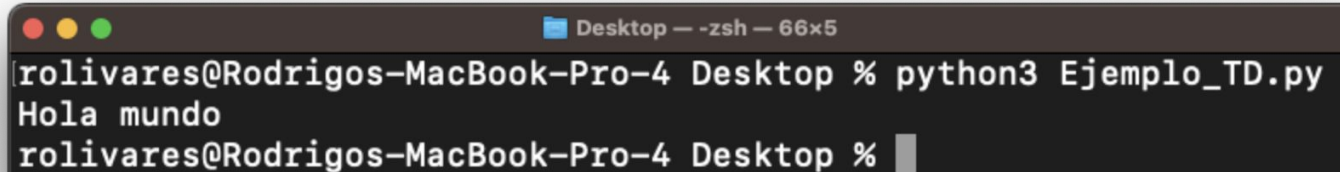
```
Desktop — zsh — 68x5
rolivares@Rodrigos-MacBook-Pro-4 Desktop % python3 Ejemplo_TD.py
<class 'str'>
rolivares@Rodrigos-MacBook-Pro-4 Desktop %
```

```
n = 2 + 3  
print(n)
```



```
Desktop — zsh — 66x5  
rolivares@Rodrigos-MacBook-Pro-4 Desktop % python3 Ejemplo_TD.py  
5  
rolivares@Rodrigos-MacBook-Pro-4 Desktop %
```

```
s = "Hola " + "mundo"  
print(s)
```

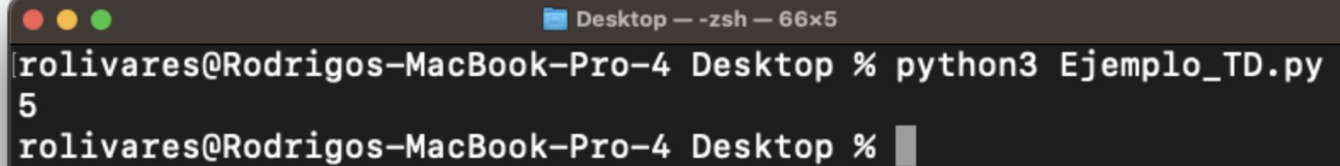


```
Desktop — zsh — 66x5  
rolivares@Rodrigos-MacBook-Pro-4 Desktop % python3 Ejemplo_TD.py  
Hola mundo  
rolivares@Rodrigos-MacBook-Pro-4 Desktop %
```

Reglas de precedencia

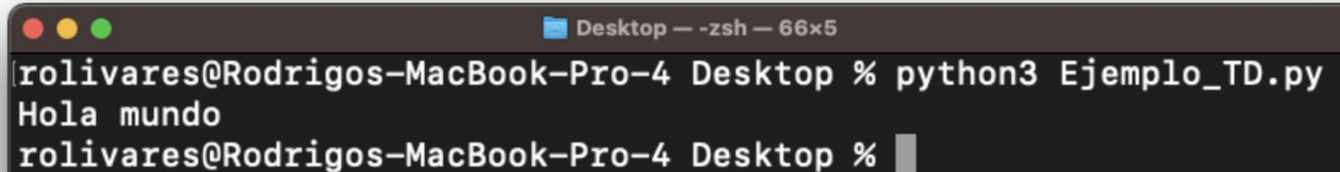
concatenar = juntar o unir

```
n = 2 + 3  
print(n)
```



```
Desktop — zsh — 66x5  
rolivares@Rodrigos-MacBook-Pro-4 Desktop % python3 Ejemplo_TD.py  
5  
rolivares@Rodrigos-MacBook-Pro-4 Desktop %
```

```
s = "Hola " + "mundo"  
print(s)
```

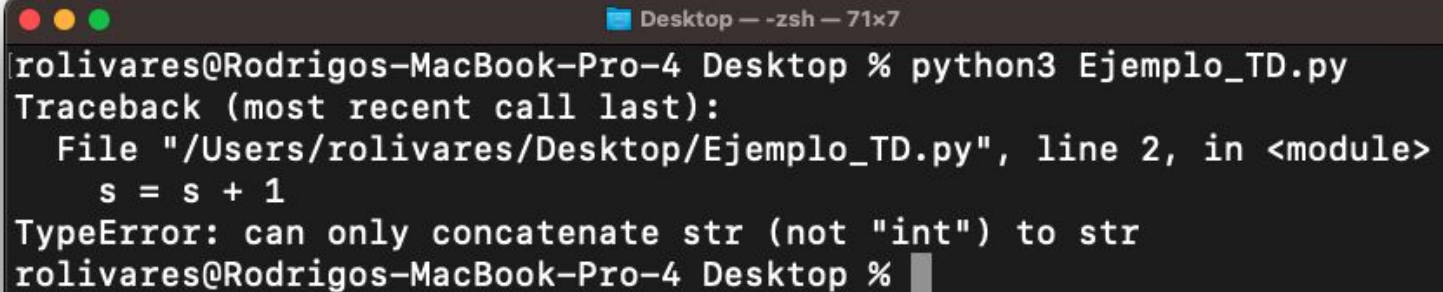


```
Desktop — zsh — 66x5  
rolivares@Rodrigos-MacBook-Pro-4 Desktop % python3 Ejemplo_TD.py  
Hola mundo  
rolivares@Rodrigos-MacBook-Pro-4 Desktop %
```

Reglas de precedencia

concatenar = juntar o unir


```
s = "Hola " + "mundo"  
s = s + 1  
print(s)
```



Desktop — zsh — 71x7

```
rolivares@Rodrigos-MacBook-Pro-4 Desktop % python3 Ejemplo_TD.py  
Traceback (most recent call last):  
  File "/Users/rolivares/Desktop/Ejemplo_TD.py", line 2, in <module>  
    s = s + 1  
TypeError: can only concatenate str (not "int") to str  
rolivares@Rodrigos-MacBook-Pro-4 Desktop %
```



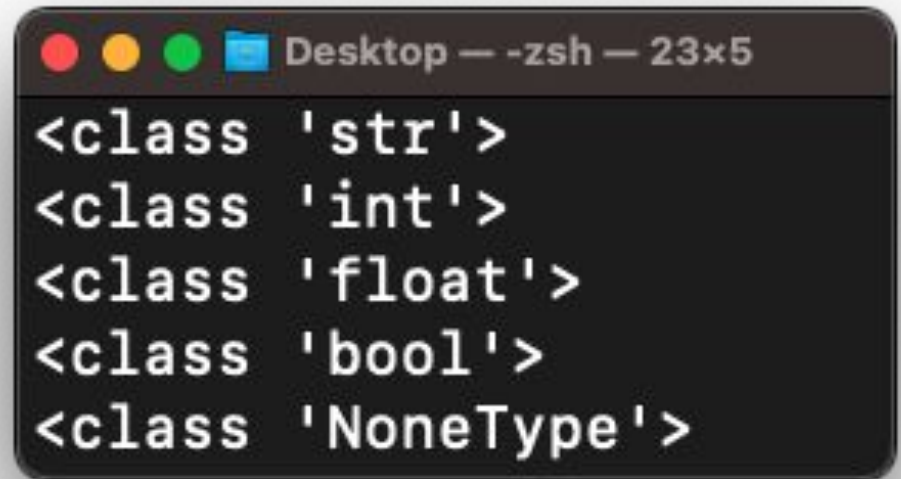
```
cadena = "texto"  
print(type(cadena))
```

```
numEntero = 1  
print(type(numEntero))
```

```
numReal = 1.0  
print(type(numReal))
```

```
booleano = True  
print(type(booleano))
```

```
vacio = None  
print(type(vacio))
```



Desktop — -zsh — 23x5

```
<class 'str'>  
<class 'int'>  
<class 'float'>  
<class 'bool'>  
<class 'NoneType'>
```

Y otros más ...

Variedad de **tipos** de números

- Los números tienen dos tipos principales.
 - Enteros:
 - **-14, -2, 0, 1, 100, 401233.**
 - Números de punto flotante: entero y decimal:
 - **-2.5, 0.0, -98.6, 14.0. Obs:** Usar punto (.) para separar la parte entera, de la parte decimal.

Conversiones de **tipo**

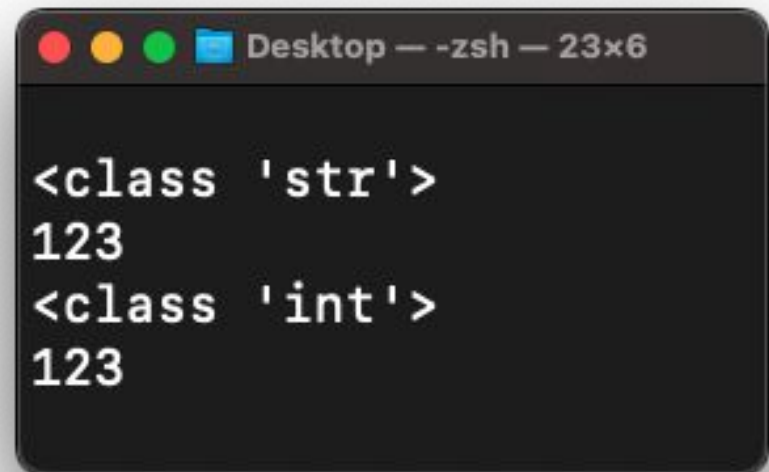
- Cuando coloca un entero y un flotante en una expresión, el entero es convertido implícitamente a flotante.
- Es posible controlar esta conversión con las funciones incorporadas ***int()*** y ***float()***.

Conversiones de **cadenas**

- Es posible usar *int()* y *float()* para convertir entre cadenas y enteros. Obtendrá un **error** si la cadena **no contiene solo** caracteres numéricos.

```
sVar = "123"  
print(type(sVar))  
print(sVar)
```

```
nVar = int(sVar)  
print(type(nVar))  
print(nVar)
```



```
Desktop - -zsh - 23x6  
  
<class 'str'>  
123  
<class 'int'>  
123
```

Entrada de datos

Entrada de datos del usuario

- Podemos instruir a Python que haga una pausa y lea datos directamente del usuario a través de la función `input()`. La función `input()` regresa una cadena.

```
print("¿Cuál es tu nombre?")  
nombre = input()  
print("Hola", nombre)
```

Convertir entrada de datos del usuario

- Si se desea **leer** un número, se debe **convertir** de cadena a número, usando una función de conversión de tipo.

```
edad = input("¿Cuál es tu edad?")  
edad = int(edad) + 1  
print("Vas a cumplir", edad, "años")
```

try / except

Manejo de excepciones

Las líneas de código con **try** y **except** permiten evitar problemas.

Si la línea de código **try** funciona, el programa omitirá la línea de **except**

Si la línea de código **try** falla, el programa ejecuta la línea de **except**

Try / except

```
var = "Hola Juan"
var = int(var)
print ("Primero", var)

var2 = "123"
var2 = int(var2)
print ("Segundo", var2)
```

```
Traceback (most recent call last):
  File "notry.py", line 2, in <module>
    istr = int(astr)
ValueError: invalid literal for int() with base 10: 'Hola Juan'
```

Modo de uso

```
entrada = input("Ingresa un numero: ")
try:
    ival = int(entrada)
except:
    print ("[ERROR]: No es un numero")
```

Con este fragmento de código, el programa no arrojará un error si el usuario ingresa un elemento diferente al esperado.

Generalidades

Comentarios

- Cualquier instrucción después del signo # es ignorado por Python (**en la misma línea**).
- **¿Por qué comentar?**
 - Describe lo que pasará en una secuencia de código.
 - Documenta quién escribió el código u otra información secundaria.
 - Desactiva una línea de código - quizás temporalmente.

Operaciones con cadenas

- Algunos operadores que se aplican a las cadenas:
 - `+` implica “concatenación”.
 - `*` implica “concatenación múltiple”.
- Python invoca internamente la función `type()` para saber cuando trabaja con una cadena o un número.

```
print('123' + "abc")
```

```
print('hola' * 5)
```

Nombres **Mnemónicos** de variables

- Las variables se deben nombrar de manera que nos ayuden a recordar lo que pretendemos almacenar en ellas (“**mnemónico**” = “**ayuda a la memoria**”).

```
x1q3z9ocd = 35.0  
x1q3z9afd = 12.50  
x1q3p9afd = x1q3z9ocd + x1q3z9afd  
print(x1q3p9afd)
```

```
a = 35.0  
b = 12.50  
c = a + b  
print(c)
```

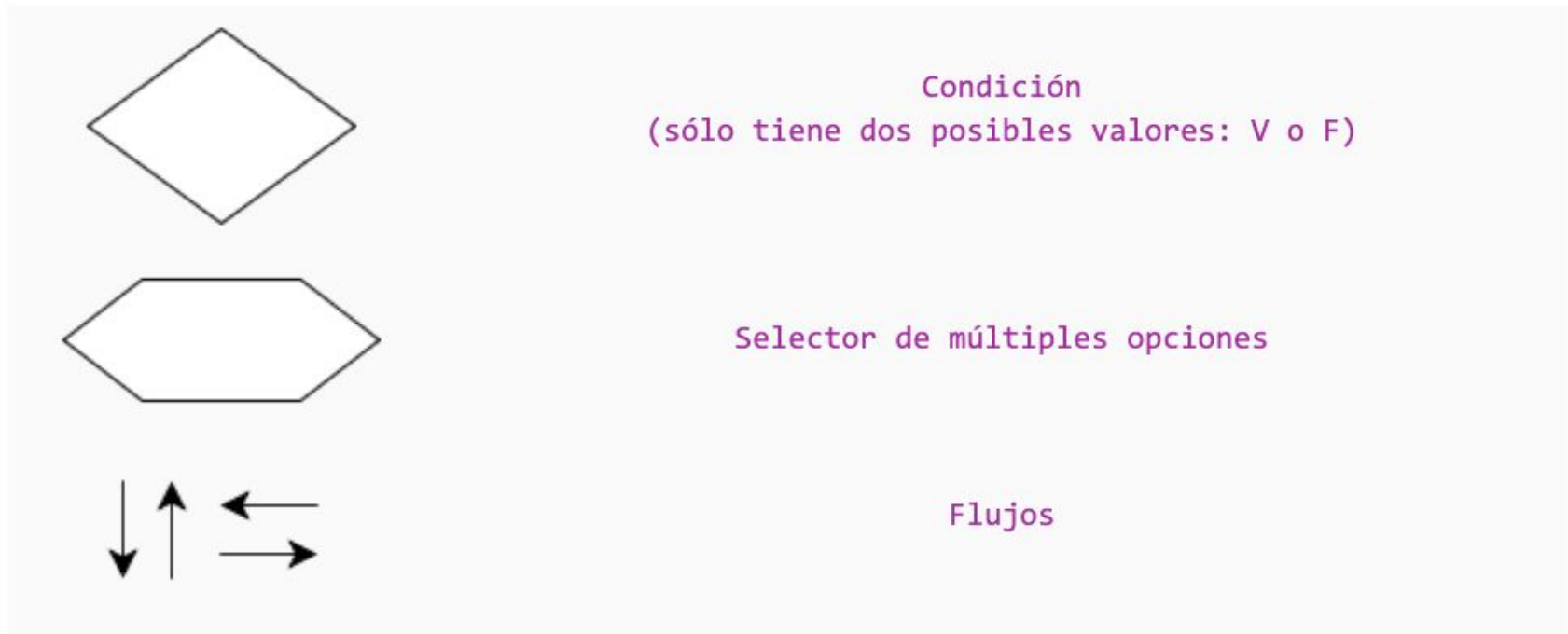
Medios de representación

Diagramas de flujo

Para la representación de soluciones, utilizando **Diagrama de Flujos**, debemos conocer la simbología:

Símbolo	Representación
	Inicio o fin del diagrama
	Entrada de datos (lectura, interacción de entrada)
	Operación/Expresión
	Mensaje (escritura, interacción de salida)

Para la representación de soluciones, utilizando **Diagrama de Flujos**, debemos conocer la simbología:



Ejercicio

Ejercicio

- Escriba un programa que pida al **usuario las horas**, para calcular el **pago bruto y líquido (80% del bruto)**. El valor de las horas es fijo, e igual a 1600.
- Por ejemplo

Entrada:

- Ingrese la cantidad de horas: **44**
- El valor de la hora es fijo: **1600**

Código en:

bit.ly/colab116

Salida del programa:

- El pago bruto es: **70400**
- El pago líquido es: **56320**



Solución (Diagrama de Flujo y código en Python3)

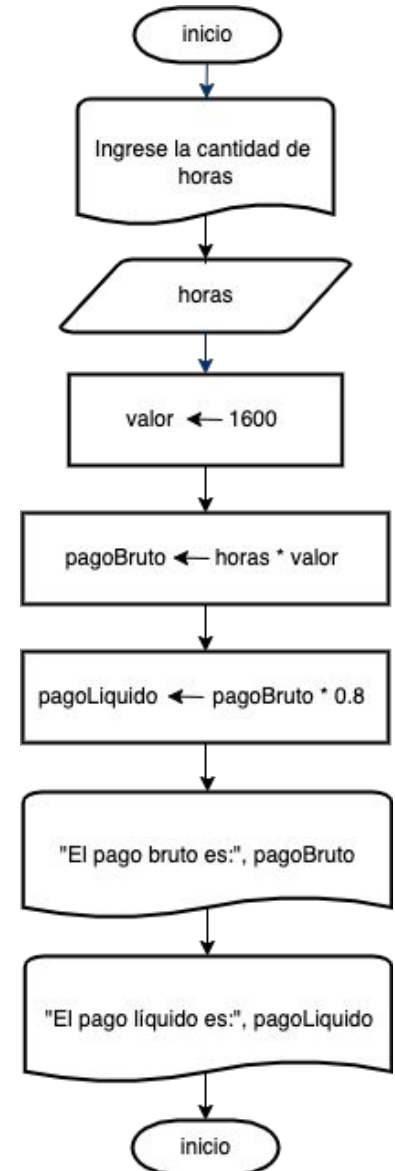
```
print("Ingrese la cantidad de horas")  
horas = int(input())
```

```
valorHora = 1600
```

```
pagoBruto = horas * valorHora
```

```
pagoLiquido = int(pagoBruto * 0.8)
```

```
print("El pago bruto es: ", pagoBruto)  
print("El pago líquido es: ", pagoLiquido)
```



Ejercicio

Escriba un programa que pida al **usuario ingresar dos valores y los almacene en dos variables**, para calcular la **suma**, la **resta**, la **multiplicación**, la **división real**, la **división entera**, el **módulo**, el **exponente** y la **raíz cuadrada** de cada valor.

Código en:

bit.ly/colab116

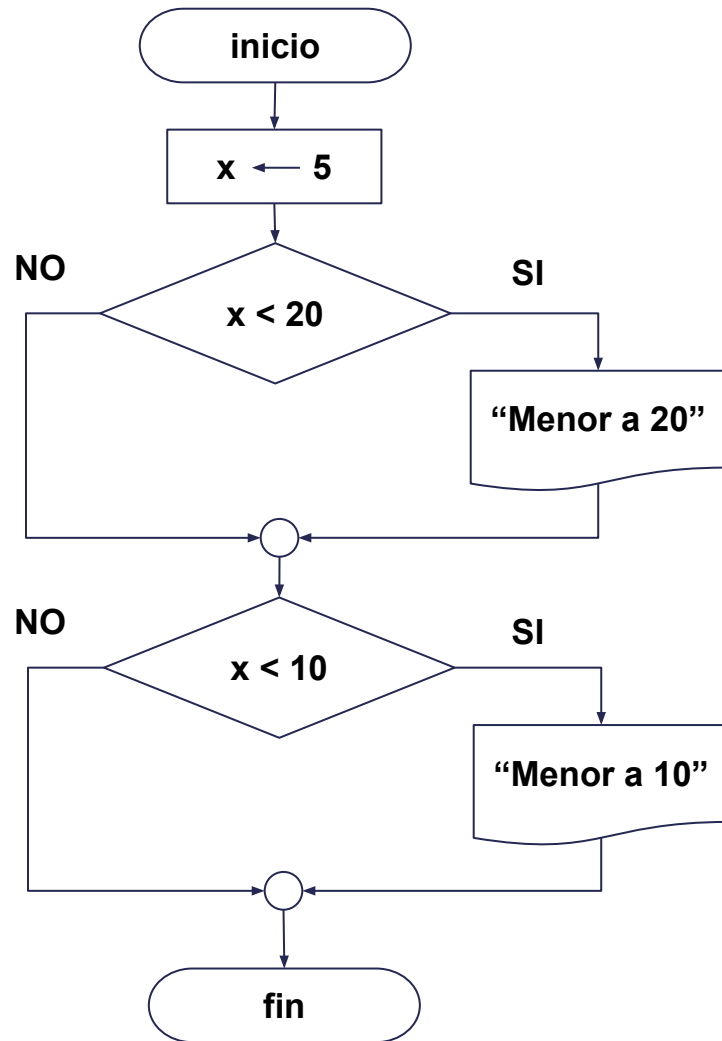
Además, mostrar el valor de verdad de:

Mayor que, Menor que, mayor o igual que, menor o igual que, igualdad y desigualdad.



Estructuras condicionales

Estructuras selectivas - Simples



$x = 50$

```
if (x < 20):  
    print ("Menor a 20")  
    print ("Chao")
```

```
if x < 10:  
    print ("Menor a 10")
```

- Las expresiones booleanas **hacen una pregunta** y generan un resultado de **Sí (True)** o **No (False)** que usamos para controlar el flujo del programa.
- Las expresiones booleanas que usan operadores de **comparación**. Evalúan el grado - **Verdadero / Falso** (Sí o No).
- Los operadores de comparación tienen en cuenta las variables, pero **no** las **cambian**.
- **RECUERDE:** Use “**Tab**” o espacios (no recomendado) para las instrucciones bajo la condición (**indentación**).

```
if (x == 5) :  
    print ("Igual a 5")  
if x > 4 :  
    print ("Mayor a 4")  
if x >= 5 :  
    print ("Mayor o igual a 5")  
if x < 6 : print ("Menor a 6")  
if x <= 5 :  
    print ("Menor o igual a 5")  
if x != 6 :  
    print ("Diferente a 6")
```

- **Aumentar** la indentación después de las instrucciones **if**, **elif**, **else**, **while**, **for...** (después de :).
- **Mantener** la **indentación** para indicar la amplitud del bloque (aquellas líneas que se verán afectadas por **if**, **elif**, **else**, **while**, **for**, ...).
- **Reducir** la indentación para volver al nivel de las líneas de las instrucciones previas, indicando así el final del bloque.
- El programa ignorará las **líneas en blanco**, pues no afectan a la indentación.
- Respecto a la indentación, los **comentarios** en las líneas se ignoran.

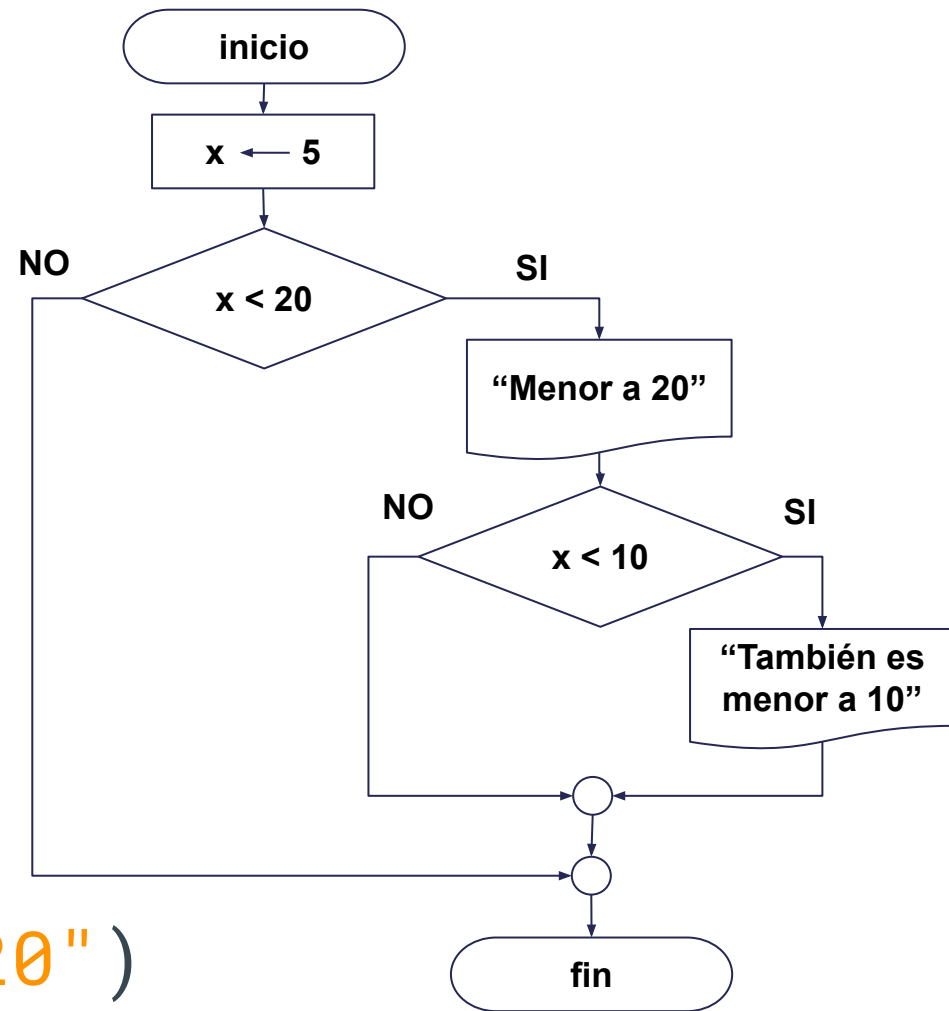
x = 5

```
if (x < 20):
```

```
    print ("Menor a 20")
```

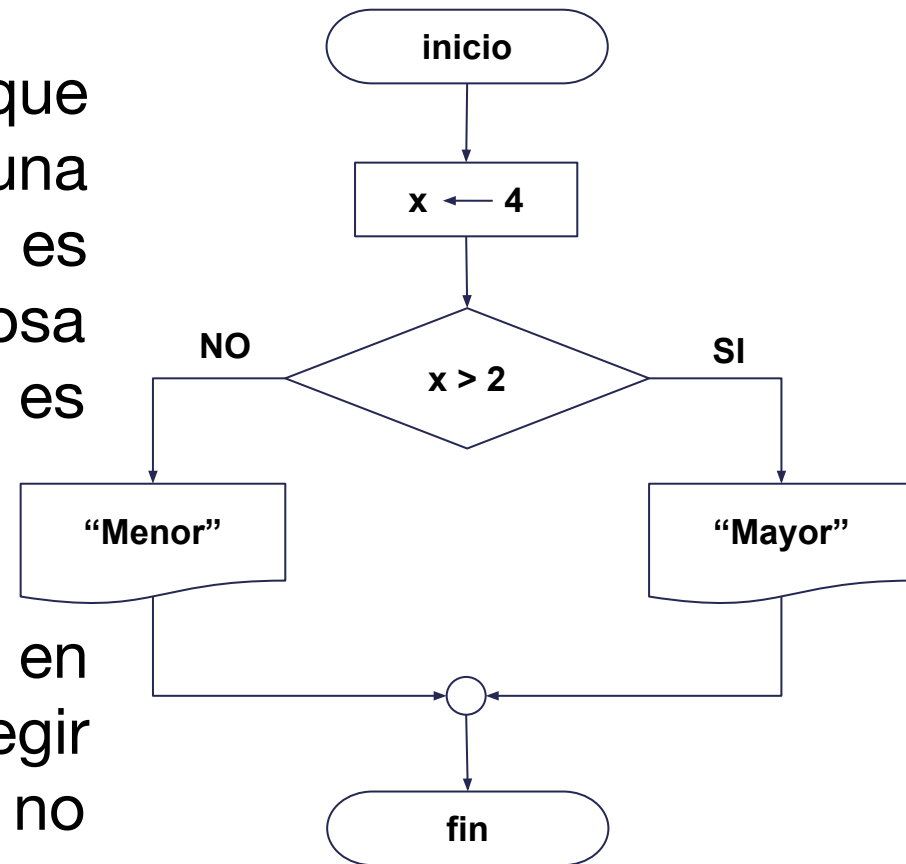
```
    if x < 10:
```

```
        print ("También es menor a 20")
```



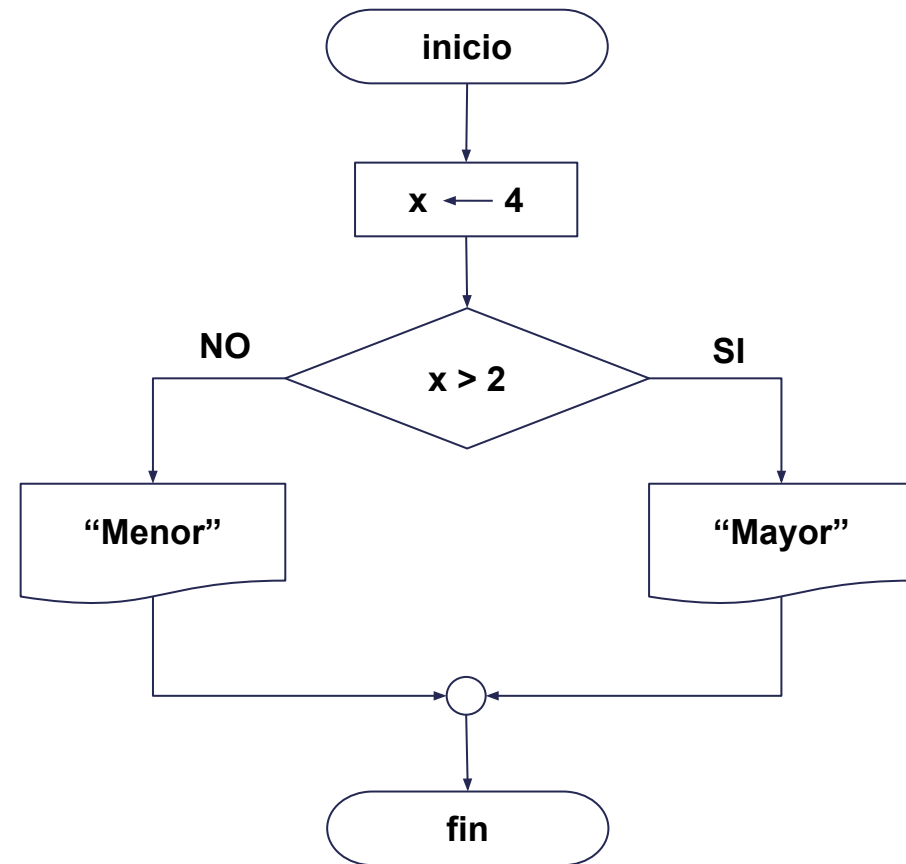
Estructuras selectivas - Dobles

- En ocasiones queremos que ocurra una cosa si una expresión lógica es verdadera y otra cosa diferente si la expresión es falsa.
- Es como una **bifurcación** en el camino: se puede elegir una u otra dirección, pero no ambas.



$x = 4$

```
if x > 2:  
    print ("Mayor")  
else:  
    print ("Menor")  
print ()
```



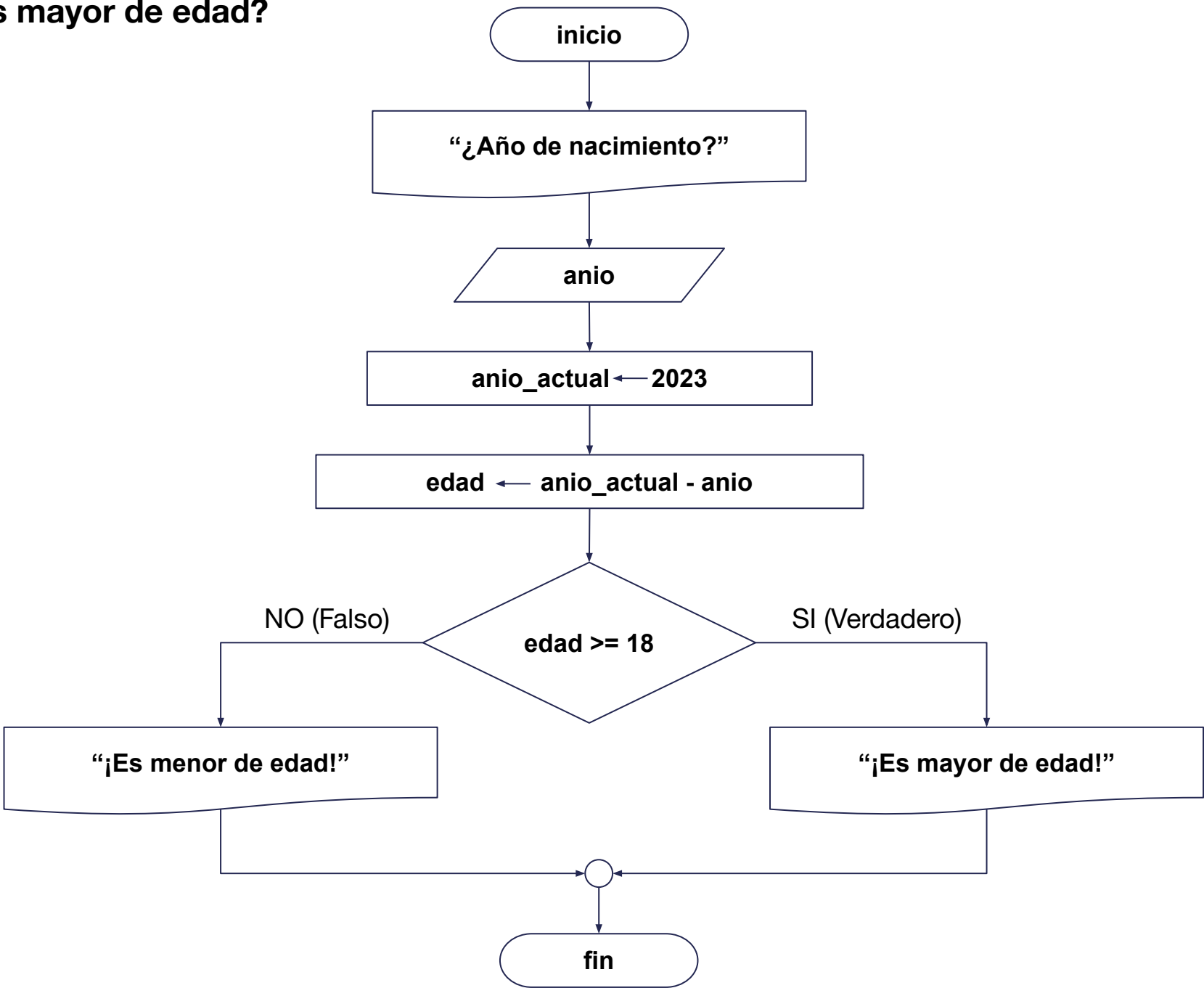
Ejercicio

¿Eres mayor de edad?

Para calcular la edad aproximada de una persona (en años), se realiza una expresión aritmética simple: resta entre el año actual y el año de nacimiento.

- (a) Realice un Diagrama de flujo que lea el año de nacimiento de una persona e indique si ésta es o no mayor de edad.
- (b) Luego, implemente la solución en Python 3.

¿Eres mayor de edad?



¿Eres mayor de edad?

```
print("¿Año de nacimiento?")
```

```
anio = int(input())
```

```
anio_actual = 2023
```

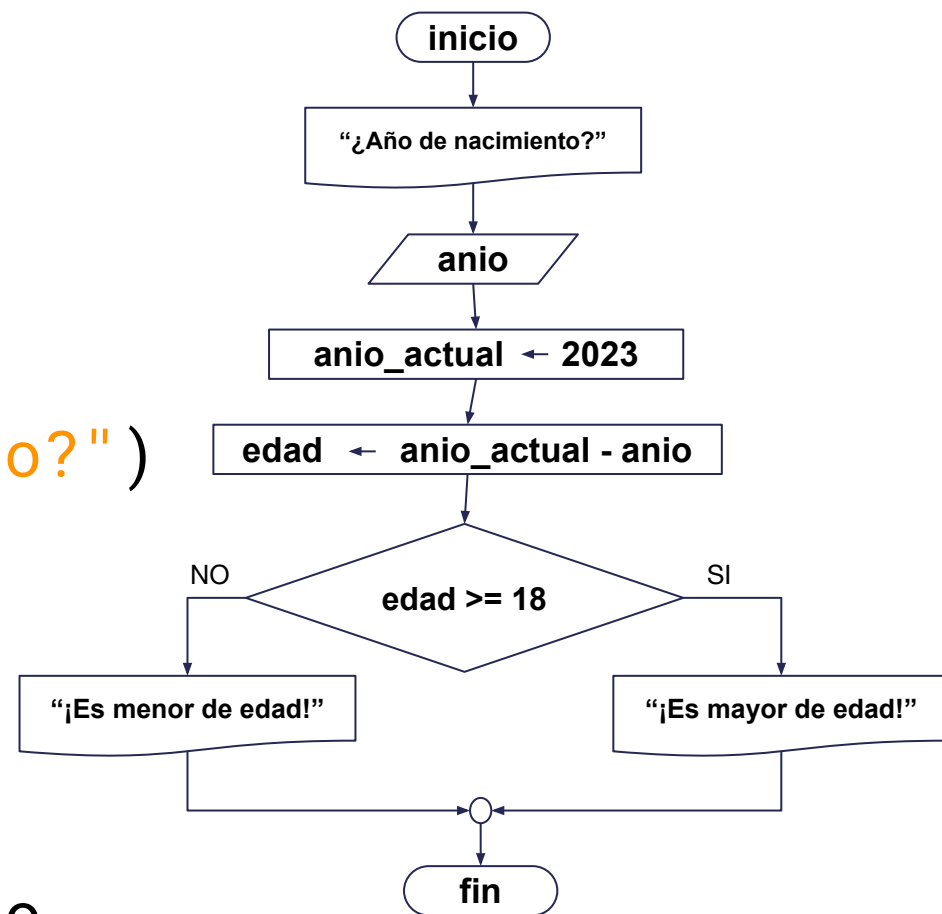
```
edad = anio_actual - anio
```

```
if edad >= 18:
```

```
    print("¡Es mayor de edad!")
```

```
else:
```

```
    print("¡Es menor de edad!")
```



Edad exacta (años, meses y días)

Calcule la edad exacta de una persona, es dinámica y cambia día tras día.

- (a) Realice un Diagrama de flujo que lea el año, el mes y el día de nacimiento de una persona e indique su edad exacta. Por simplicidad, considere que todos los meses tienen 30 días.
- (b) Luego, implemente la solución en Python 3. Posibles salidas considerando la fecha actual como 13 de abril de 2023:

```
Ingrese el dia de nacimiento: 20
Ingrese el mes de nacimiento: 4
Ingrese el año de nacimiento: 2018
```

```
La edad exacta es:
4 año(s) 11 mes(es) 23 día(s)
```

```
Ingrese el dia de nacimiento: 7
Ingrese el mes de nacimiento: 10
Ingrese el año de nacimiento: 2008
```

```
La edad exacta es:
14 año(s) 6 mes(es) 6 día(s)
```

```
Ingrese el dia de nacimiento: 10
Ingrese el mes de nacimiento: 4
Ingrese el año de nacimiento: 2018
```

```
La edad exacta es:
5 año(s) 0 mes(es) 3 día(s)
```

```
Ingrese el dia de nacimiento: 13
Ingrese el mes de nacimiento: 12
Ingrese el año de nacimiento: 1999
```

```
La edad exacta es:
23 año(s) 4 mes(es) 0 día(s)
```

```
dia_nac = int(input("Ingrese el dia de nacimiento: "))
mes_nac = int(input("Ingrese el mes de nacimiento: "))
anio_nac = int(input("Ingrese el año de nacimiento: "))

dia_actual = 13
mes_actual = 4
anio_actual = 2023

anio_exac = anio_actual - anio_nac
mes_exac = mes_actual - mes_nac
dia_exac = dia_actual - diaNac

if dia_exac < 0:
    dia_exac = dia_exac + 30
    mes_exac = mes_exac - 1

if mes_exac < 0:
    mes_exac = mes_exac + 12
    anio_exac = anio_exac - 1

print("La edad exacta es:", anio_exac, "año(s)",
mes_exac, "mes(es)", dia_exac, "día(s)")
```

¿Cuál es el Diagrama de Flujo?

Año bisiesto

Un año es bisiesto si es múltiplo de 4, exceptuando los múltiplos de 100, que sólo son bisiestos cuando son múltiplos además de 400. Por ejemplo el año 1400 no fue bisiesto, pero el año 2016 sí lo fue. El año 2023 no es bisiesto, al igual que no lo será el 2100. Sin embargo, el 2040, si será bisiesto.

- (a) Desarrolle el diagrama de flujo que solicite un año e indique si fue, es o será bisiesto, o lo contrario (no fue, no es o no será).
- (b) Implemente la solución en Python 3. Algunas posibles salidas:

```
Ingrese el año: 2018
El año 2018 NO fue bisiesto
```

```
Ingrese el año: 2016
El año 2016 SI fue bisiesto
```

```
Ingrese el año: 1400
El año 1400 NO fue bisiesto
```

```
Ingrese el año: 2023
El año 2020 NO es bisiesto
```

```
Ingrese el año: 2100
El año 2100 NO será bisiesto
```

```
Ingrese el año: 2048
El año 2048 SI será bisiesto
```



```
anio = int(input("Ingrese el año: "))
anio_actual = 2023
if anio < anio_actual:
    verbo = "fue"
else:
    if anio > anio_actual:
        verbo = "será"
    else:
        verbo = "es"

if anio < 1582:
    print("El año", anio, "NO", verbo, "bisiesto")
else:
    resto = anio % 4
    if resto == 0:
        resto = anio % 100
        if resto == 0:
            resto = anio % 400
            if resto == 0:
                print("El año", anio, "SI", verbo, "bisiesto")
            else:
                print("El año", anio, "NO", verbo, "bisiesto")
        else:
            print("El año", anio, "SI", verbo, "bisiesto")
    else:
        print("El año", anio, "NO", verbo, "bisiesto")
else:
    print("El año", anio, "NO", verbo, "bisiesto")
```

¿Cuál es el Diagrama de Flujo?

Estructuras selectivas - Múltiples

```
if x < 2:  
    print ("pequeno")  
elif x < 10:  
    print ("MEDiano")  
else:  
    print ("GRANDE")
```

¿Cuál es el
Diagrama de
Flujo?

```
if x < 2:
    print ("Pequeno")
elif x < 10:
    print ("Mediano")
elif x < 20:
    print ("Grande")
elif x < 40:
    print ("Largo")
elif x < 100:
    print ("Muy largo")
else:
    print ("Gigante")
```

Cuál nunca se imprimirá?

```
if x < 2 :  
    print ( "menor a 2" )  
elif x >= 2 :  
    print ( "dos o mas" )  
else :  
    print ( "alguna cosa" )
```

Ejercicios

Desarrolle un diagrama de flujo que pida al usuario un número entero y muestre por pantalla si es par o impar. Luego, escriba el programa en Python 3.

Desarrolle un diagrama de flujo que le pida al usuario ingresar un día de la semana (número) e imprima un la glosa (lunes, martes, miércoles, ..., domingo). Si el día ingresado no corresponde, el programa debe imprimir un mensaje de error. Luego, escriba el programa en Python 3.

Para tributar un determinado impuesto se debe ser mayor de 18 años y tener ingresos iguales o superiores a 1 millón mensual. Desarrolle un diagrama de flujo que pregunte al usuario su edad y su ingreso mensual, y luego muestre por pantalla si el usuario tiene que tributar o no. Luego, escriba el programa en Python 3.

Ejercicios

Desarrolle un diagrama de flujo que solicite al usuario una letra y, si es una vocal, muestre el mensaje “es vocal”, de lo contrario no muestra nada. Luego, escriba el programa en Python 3.

Una empresa tiene salas de juegos para todas las edades y quiere calcular de forma automática el precio que debe cobrar a sus clientes. Desarrolle un diagrama de flujo que pregunte al usuario la edad del cliente y muestre el valor de la entrada. Si el cliente es menor de 4 años, puede entrar gratis, si tiene entre 4 y 18 años debe pagar 5 mil y si es mayor de 18 años, paga 10 mil. Luego, escriba el programa en Python 3.

Ejercicios

Los tramos impositivos para la declaración de la renta son los siguientes:

- (i) Menos de 500 mil, el impuesto es de 5%.
- (ii) Entre 500 mil y 1 millón, el impuesto es de 15%.
- (iii) Entre 1 millón y 1 millón 500 mil, el impuesto es de 20%
- (iv) Entre 1 millón 500 mil y 2 millones 500 mil, el impuesto es de 30%
- (v) Más de 2 millones 500 mil, el impuesto es de 45%.

Desarrolle un diagrama de flujo que le pida al usuario su renta anual y muestre por pantalla el impuesto que le corresponde. Luego, escriba el programa en Python 3.

Fundamentos de Programación

Prof. Dr. Roberto Muñoz S. (Sec 1, Lab 502)
Prof. Dr. Rodrigo Olivares O. (Sec 2, Lab 401)
Prof. Víctor Ríos (Sec 3, Lab 303)
Prof. Pablo Olivares (Sec 4, Lab 301)

Escuela de Ingeniería en Informática
Facultad de Ingeniería
Universidad de Valparaíso