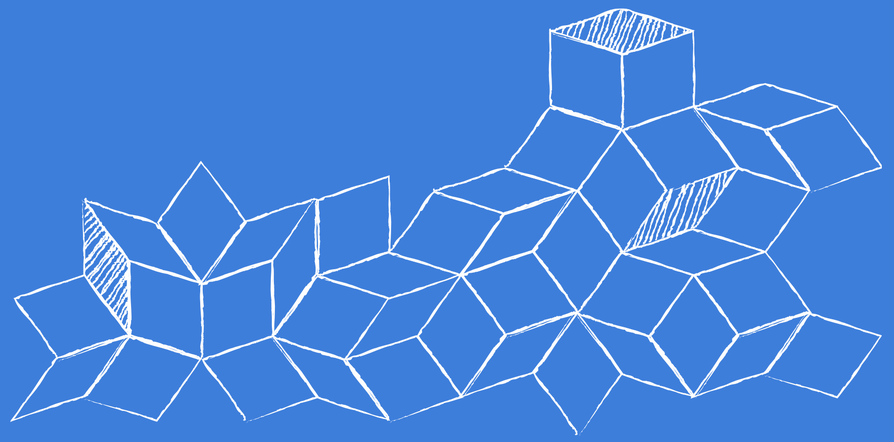
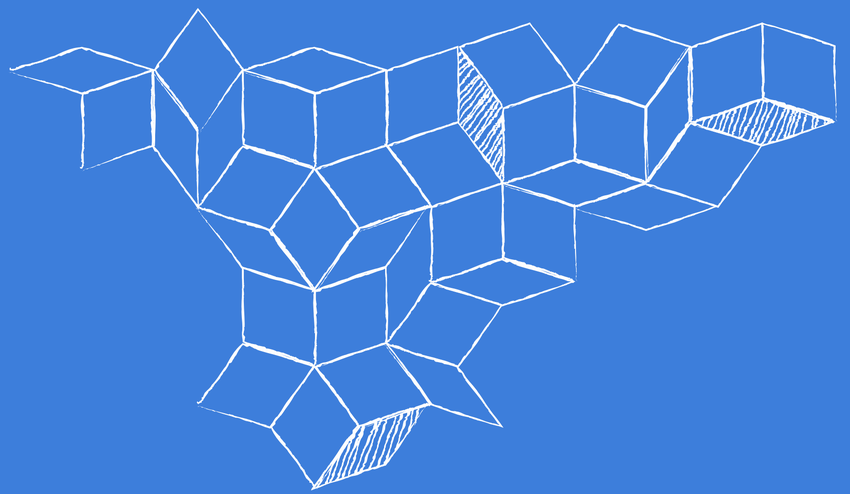


# STEMUC

## Academia para **escolares**



Suma MAT UC

preingeniería


PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

PRIMEROS PASOS
EN PROGRAMACIÓN

Clase 2

# Instrucciones condicionales

Antes de partir…

**Repasemos un poco lo que vimos ayer**

# Algoritmos

## Pensamiento Computacional

1. Descomponer
   - Dividir el problema en partes más pequeñas y manejables.
2. Reconocer patrones
   - Buscar similitudes entre las distintas partes y problemas que ya han solucionado.
3. Abstraerse
   - Concentrarse en lo importante e ignorar lo irrelevante.
4. Algoritmo
   - Diseñar pasos o reglas que solucionen cada parte del problema.

# Expresiones

Todos los programas que escribiremos se pueden construir con:

- Datos de **entrada** (*input*) que se leen.
- **Variables** que recuerdan datos.
- Instrucciones **condicionales** que se ejecutan dependiendo de una condición.
- Datos de **salida** (*output*) que se escriben.
- **Operaciones matemáticas** que calculan datos.
- Instrucciones **repetitivas** que se ejecutan múltiples veces dependiendo de una condición.

# En detalle aprendieron sobre…

- Distintos tipos de datos: *int*, *float*, *str*, *bool*.
- Operadores asignación (=), aritméticos (+, -, /, …), comparación (==, >, <=, …) y lógicos (and, or, not).
- Sintaxis print:

```
print(variable1, variable2, variable3)
print('mensaje1' + 'mensaje2', variable1)
```

- Sintaxis input:

```
variable = input(mensaje)
```

# En detalle aprendieron sobre…

Veamos un detalle de la instrucción `print()`.

```
print(variable1, variable2, variable3, ...)
print(str1 + str2 + str3 + ...)
```

# En detalle aprendieron sobre...

```
sopaipillas = 0
empanadas = 0
sopaipillas = sopaipillas + 1
empanadas = empanadas + 2
print("Me da", sopaipillas,"sopaipilla y", empanadas,"empanadas, por favor.")
```

```
Me da 1 sopaipilla y 2 empanadas, por favor.
```

# En detalle aprendieron sobre…

```
sopaipillas = 0
empanadas = 0
sopaipillas = sopaipillas + 1
empanadas = empanadas + 2
print("Me da" + sopaipillas + "sopaipilla y" + empanadas +
      "empanadas, por favor.")
```

```
Traceback (most recent call last):
  File “x", line 5, in <module>
    print("Me da" + sopaipillas + "sopaipilla y" + empanadas +
          "empanadas por favor.")
TypeError: can only concatenate str (not "int") to str
```

# En detalle aprendieron sobre…

```
sopaipillas = 0
empanadas = 0
sopaipillas = sopaipillas + 1
empanadas = empanadas + 2
print("Me da " + str(sopaipillas) + “ sopaipilla y " + str(empanadas) +
      “ empanadas, por favor.")
```

```
Me da 1 sopaipilla y 2 empanadas, por favor.
```

**Nota:** Si vamos a concatenar, debemos convertir el número a *string* y utilisar el espaciamiento adecuado.

# En detalle aprendieron sobre…

Veamos un detalle de la instrucción `input()`.

Todas las entradas pedidas con `input()` serán siempre *string*, por lo que será necesario casterar los datos para poder utilizarlos correctamente.

```
# Pido input al usuario y lo guardo en a
a = input("Ingrese un str: ")
# Pido un int al usuario
b = int(input("Ingrese un int: ") )
# Pido un float al usuario
c = float(input("Ingrese un float: "))
# Pido un bool al usuario
d = bool(input("Ingrese un bool: "))
print(a, b, c, d)
```

**Hagamos un ejercicio**

para calentar la materia

# Problemas

- Escribe un programa que permita calcular el promedio de 3 pruebas.
- Evaluar el polinomio $x^4 + 12x^3 + 2x^2 - x$ para un $x$ entregado por el usuario.
- Obtener la unidad de una variable $x$ entregada por el usuario.
  Por ejemplo: si $x = 1234$, se obtendrá $4$.
- Pide un número $x$, un número $y$, e imprime el $x$ -ésimo dígito de $y$.
- Si Pepa se demora $m$ minutos en recorrer un kilómetro, ¿a qué velocidad se está moviendo en $km/h$ ?

Ahora sí, ¡vemos materia nueva!

**Instrucciones condicionales**

# En la clase de hoy…

- Aprenderemos las nuevas instrucciones `if`, `elif`, `else`.
- Con ellos crearemos **flujos condicionales**.

# Condicionales

Instrucciones condicionales:

- Permiten ejecutar una o más instrucciones **solamente** si se cumple una condición.

Una condición es:

- Expresión cuyo valor es del tipo *`bool`*.
- Por lo mismo, solo tiene dos valores posibles: `True` o `False`.

# Operadores comparativos

Estos operadores se utilizan para comparar dos valores, retornando como resultado de esa comparación un valor del tipo *`bool`*.

| Operador | Descripción |
|---|---|
| `a == b` | Retorna `True` ssi a es **igual** a `b`. |
| `a != b` | Retorna `True` ssi a es **distinto** a `b`. |
| `a < b` | Retorna `True` ssi a es **menor** a `b`. |
| `a > b` | Retorna `True` ssi a es **mayor** a `b`. |
| `a <= b` | Retorna `True` ssi a es **menor o igual** a `b`. |
| `a >= b` | Retorna `True` ssi a es **mayor o igual** a `b`. |

# Operadores lógicos

Estos operadores se utilizan para comparar dos valores, retornando como resultado de esa comparación un valor del tipo *`bool`*.

| Operador | Descripción |
|---|---|
| `a and b` | Retorna `True` ssi a **y** b son `True`. |
| `a or b` | Retorna `True` ssi a **o** b son `True`. |
| `not a` | Retorna `True` ssi a **es** **`False`**. |

# Control de flujo: `if`, `elif`, `else`

Hablaremos de la ejecución de código opcional, en donde algo ocurre si se cumplen las condiciones que especificamos en el programa. Para programar bien las condiciones, será útil conseguir un manejo medianamente bueno de lógica.

- ¿Cómo podemos implementar un programa que pida *inputs* al usuario, y dependiendo de su respuesta ejecute soluciones distintas?
- ¿Qué cosas simples podríamos hacer de forma condicional?
- ¿Cómo nos ayudan los operadores condicionales y los operadores lógicos?

# if

La instrucción `if` permite ejecutar una sección de código **si** se cumple una condición.

```
if condición:          # Se abre la condición
    código_if          # Se ejecuta SOLO si se cumple la condición
    .
    .
    .

código_fuera_de_if  # Se ejecuta cuando se sale del if
```

**Pregunta:** Es bien distinta esta sintaxis, ¿o no? ¿Qué tiene de nuevo?

# `if`

Hay que tener cuidado con la indentación en las instrucciones dentro de la instrucción `if`, estas deben tener exactamente la misma indentación.

Si lo anterior no se cumple, el intérprete de Python levantará un error.

# if

La condición a la que alude el ejemplo se refiere a cualquier expresión que evalúe a `True` o `False`. Puede ser una variable lógica u operaciones de comparación.

```
nombre = input("Ingrese su nombre: ")

if nombre == "Pepa":
    print("Hola, te estaba esperando")

print("Gracias")
```

# if

También podemos hacer que la condición sea una variable, y usarla para verificar en la instrucción.

```
nombre = input("Ingrese su planeta: ")
planeta_federado = "Andoria"

if nombre == planeta_federado:
    print("Puede cruzar la frontera")
```

# if, else

La instrucción `else` permite ejecutar una sección de código **si no** se cumple una o más condiciones anteriores.

```
if condición:                  # Se abre la condición
    código_if                  # Se ejecuta SOLO si se cumple la condición
    .
    .
    .

else:
    código_else                # Se ejecuta si la condición resulta False
    .
    .
    .

código_fuera_de_if_else   # Se ejecuta cuando se sale del if/else
```

# if, else

Podríamos entonces expandir los ejemplos anteriores, ejecutando una sección de código en caso de que no ocurra la condición.

```
nombre = input("Ingrese su nombre: ")

if nombre == "Pepa":
    print("Hola, te estaba esperando")
else:
    print("Qué onda?! No eres Pepa!!!")

print("Gracias")
```

# if, else

```
número = int(input("Ingresa un número del 0 al 9: "))

if numero <= 9:
    print("Bien! El número es", numero)
else:
    print("No de nuevo... Ingresa un número del 0 al 9.")
```

**Pregunta:** ¿Será suficiente? ¿Qué necesitaríamos para robustecer esta solución al problema?

# if, elif, else

La instrucción `elif` permite ejecutar una sección de código si se cumple una condición y no se ha cumplido ningún `if` o `else` anterior.

```
if condición_1:          # Se abre la condición
    código_if
    .
    .
elif condición_2:        # Si if es False y elif es True
    código_elif
    .
    .
else:
    código_else          # Si if y elif son False
    .
    .
código_fuera_de_if_elif_else
```

# `if`, `elif`, `else`

- **Siempre** debe existir un `if`.
- Pueden haber **varios** `elif`.
- **No es obligatorio** usar `else` al final, ni poner código después del `if`/`elif`/`else`.

```
if condición_1:          # Se abre la condición
    código_if
    .
    .
elif condición_2:        # Puede haber uno
    código_elif
    .
    .
elif condición_3:
    código_else          # E incluso otro
    .
    .
# Puede que no necesitemos un else o código después
```

# if, elif, else

Volvamos al ejemplo de los números entre 0 y 9.

```
número = int(input("Ingresa un número del 0 al 9: "))
if número > 9:
    print("No de nuevo... Ingresa un número del 0 al 9.")
elif número < 0:
    print("No de nuevo... Ingresa un número del 0 al 9.")
else:
    print("Bien! El número es", número)
```

**Pregunta:** ¿Podemos aplicar los operadores lógicos para acortar este código?

# if, elif, else

Una solución simple de entender:

```
número = int(input("Ingresa un número del 0 al 9: "))
if número < 0 and número > 9:
    print("Bien! El número es", número)
else:
    print("No de nuevo... Ingresa un número del 0 al 9.")
```

Una solución rebuscada, pero posible:

```
número = int(input("Ingresa un número del 0 al 9: "))
if not(número < 0 and número > 9):
    print("No de nuevo... Ingresa un número del 0 al 9.")
else:
    print("Bien! El número es", número)
```

# Ejercicio libre 

1. En parejas, hagan un quiz sobre un tema que les interese.
2. Cuando estén listos, que otro equipo lo pruebe.

# ¿Qué es lo que aprendieron hoy?

Ve a

www.menti.com

Introduce el código

7790 9557