

# Otro Problema menos Con Python

Christian Gimenez, Jorge Rodriguez y Candelaria Alvarez

# Outline

- 1 El Problema
- 2 Listas
- 3 Tuplas
- 4 Hashes
- 5 Funciones
- 6 Muchas Gracias
- 7 Licencia



## 1 El Problema

- ¿Se Acuerdan?
- ¿Qué Otras Cosas Vimos?
- Código Anterior
- ¿Y cómo fue todo?

# ¿Se Acuerdan?

Se acerca el día de la primavera y un grupo de estudiantes querían juntarse a comer pizzas. Normalmente, sacar cuentas para poder deducir cuánto tiene que pagar cada uno sería fácil, pero querían que participen a los primeros 20 que lleguen al aula (se suponen que son los más aplicados).

## ¿Qué vimos?

- Entrada y Salida
- Variables
- Operadores matemáticos

# ¿Se Acuerdan?

Se acerca el día de la primavera y un grupo de estudiantes querían juntarse a comer pizzas. Normalmente, sacar cuentas para poder deducir cuánto tiene que pagar cada uno sería fácil, pero querían que participen a los primeros 20 que lleguen al aula (se suponen que son los más aplicados).

## ¿Qué vimos?

- Entrada y Salida
- Variables
- Operadores matemáticos

# ¿Se Acuerdan?

Se acerca el día de la primavera y un grupo de estudiantes querían juntarse a comer pizzas. Normalmente, sacar cuentas para poder deducir cuánto tiene que pagar cada uno sería fácil, pero querían que participen a los primeros 20 que lleguen al aula (se suponen que son los más aplicados).

## ¿Qué vimos?

- Entrada y Salida
- Variables
- Operadores matemáticos

# ¿Se Acuerdan?

Se acerca el día de la primavera y un grupo de estudiantes querían juntarse a comer pizzas. Normalmente, sacar cuentas para poder deducir cuánto tiene que pagar cada uno sería fácil, pero querían que participen a los primeros 20 que lleguen al aula (se suponen que son los más aplicados).

## ¿Qué vimos?

- Entrada y Salida
- Variables
- Operadores matemáticos

# El Dilema de los Palmitos

Además, los estudiantes que organizan el evento quieren ofrecer dos tipos de pizzas: con palmitos y común. Como hay varias personas que no les gustan los palmitos, deciden preguntarle a cada una si quieren con palmitos o no. Lo bueno es que no afecta al precio final y sólo debe avisarle a Juan que no le ponga palmitos.

¿Qué vimos?

- Condicionales



Además, los estudiantes que organizan el evento quieren ofrecer dos tipos de pizzas: con palmitos y común. Como hay varias personas que no les gustan los palmitos, deciden preguntarle a cada una si quieren con palmitos o no. Lo bueno es que no afecta al precio final y sólo debe avisarle a Juan que no le ponga palmitos.

¿Qué vimos?

- Condicionales

# ¡Pero son 20 estudiantes!

Ahora, no todos los estudiantes sabe cómo ejecutar el programa. Por eso, queremos repetir nuestro programa 20 veces.

¿Qué vimos?

- Repetitivas for
- `range()` o rangos

# ¡Pero son 20 estudiantes!

Ahora, no todos los estudiantes sabe cómo ejecutar el programa. Por eso, queremos repetir nuestro programa 20 veces.

## ¿Qué vimos?

- Repetitivas for
- `range()` o rangos

# ¡Pero son 20 estudiantes!

Ahora, no todos los estudiantes sabe cómo ejecutar el programa. Por eso, queremos repetir nuestro programa 20 veces.

## ¿Qué vimos?

- Repetitivas for
- `range()` o rangos

# ¿Cuántos de palmitos?

Ahora, una vez terminada la aplicación, sería ideal que Juan sepa cuántos pidieron con y sin palmitos. ¿Cómo se puede hacer esto?

¿Qué vimos?

- Variables contadoras.
- La verdadera utilidad de las repetitivas for. :)

# ¿Cuántos de palmitos?

Ahora, una vez terminada la aplicación, sería ideal que Juan sepa cuántos pidieron con y sin palmitos. ¿Cómo se puede hacer esto?

## ¿Qué vimos?

- Variables contadoras.
- La verdadera utilidad de las repetitivas for. :)

# ¿Cuántos de palmitos?

Ahora, una vez terminada la aplicación, sería ideal que Juan sepa cuántos pidieron con y sin palmitos. ¿Cómo se puede hacer esto?

## ¿Qué vimos?

- Variables contadoras.
- La verdadera utilidad de las repetitivas for. :)

## 1 El Problema

- ¿Se Acuerdan?
- ¿Qué Otras Cosas Vimos?
- Código Anterior
- ¿Y cómo fue todo?



## Tabular vs. Espaciado

- **Nunca usar el caracter tabulador.**
- Usar 4 espacios está bien.
- Mantener consistencia: siempre indentar de a 4 espacios.
- Configurar Geany para que no use tab.

## Importar tkinter (Tk)

```
from tkinter import Tk
```

## Importar la entrada y salida

- Es muy útil usar etiquetas en español.
- En vez de usar `askstring()` → usamos `pedir_texto()`

```
from tkinter.simpledialog import askstring as pedir_texto
```

## Tabular vs. Espaciado

- **Nunca usar el caracter tabulador.**
- Usar 4 espacios está bien.
- Mantener consistencia: siempre indentar de a 4 espacios.
- Configurar Geany para que no use tab.

## Importar tkinter (Tk)

```
from tkinter import Tk
```

## Importar la entrada y salida

- Es muy útil usar etiquetas en español.
- En vez de usar `askstring()` → usamos `pedir_texto()`

```
from tkinter.simpledialog import askstring as pedir_texto
```

## Tabular vs. Espaciado

- **Nunca usar el caracter tabulador.**
- Usar 4 espacios está bien.
- Mantener consistencia: siempre indentar de a 4 espacios.
- Configurar Geany para que no use tab.

## Importar tkinter (Tk)

```
from tkinter import Tk
```

## Importar la entrada y salida

- Es muy útil usar etiquetas en español.
- En vez de usar `askstring()` → usamos `pedir_texto()`

```
from tkinter.simpledialog import askstring as pedir_texto
```

## Tabular vs. Espaciado

- **Nunca usar el caracter tabulador.**
- Usar 4 espacios está bien.
- Mantener consistencia: siempre indentar de a 4 espacios.
- Configurar Geany para que no use tab.

## Importar tkinter (Tk)

```
from tkinter import Tk
```

## Importar la entrada y salida

- Es muy útil usar etiquetas en español.
- En vez de usar `askstring()` → usamos `pedir_texto()`

```
from tkinter.simpledialog import askstring as pedir_texto
```

## Tabular vs. Espaciado

- **Nunca usar el caracter tabulador.**
- Usar 4 espacios está bien.
- Mantener consistencia: siempre indentar de a 4 espacios.
- Configurar Geany para que no use tab.

## Importar tkinter (Tk)

```
from tkinter import Tk
```

## Importar la entrada y salida

- Es muy útil usar etiquetas en español.
- En vez de usar `askstring()` → usamos `pedir_texto()`

```
from tkinter.simpledialog import askstring as pedir_texto
```

## Tabular vs. Espaciado

- **Nunca usar el caracter tabulador.**
- Usar 4 espacios está bien.
- Mantener consistencia: siempre indentar de a 4 espacios.
- Configurar Geany para que no use tab.

## Importar tkinter (Tk)

```
from tkinter import Tk
```

## Importar la entrada y salida

- Es muy útil usar etiquetas en español.
- En vez de usar `askstring()` → usamos `pedir_texto()`

```
from tkinter.simpledialog import askstring as pedir_texto
```

## Tabular vs. Espaciado

- **Nunca usar el caracter tabulador.**
- Usar 4 espacios está bien.
- Mantener consistencia: siempre indentar de a 4 espacios.
- Configurar Geany para que no use tab.

## Importar tkinter (Tk)

```
from tkinter import Tk
```

## Importar la entrada y salida

- Es muy útil usar etiquetas en español.
- En vez de usar `askstring()` → usamos `pedir_texto()`

```
from tkinter.simpledialog import askstring as pedir_texto
```

## Tabular vs. Espaciado

- **Nunca usar el caracter tabulador.**
- Usar 4 espacios está bien.
- Mantener consistencia: siempre indentar de a 4 espacios.
- Configurar Geany para que no use tab.

## Importar tkinter (Tk)

```
from tkinter import Tk
```

## Importar la entrada y salida

- Es muy útil usar etiquetas en español.
- En vez de usar `askstring()` → usamos `pedir_texto()`

```
from tkinter.simpledialog import askstring as pedir_texto
```



## Probar en el momento

Podemos pedirle a Geany que **ejecute una porción de código** para ver que hace.

¡Python es interpretado!

## Zen de Python

```
import this
```

- El Zen nos dice que hagamos las cosas simples
- Muchas veces el código es:

## Probar en el momento

Podemos pedirle a Geany que **ejecute una porción de código** para ver que hace.

¡Python es interpretado!

## Zen de Python

```
import this
```

- El Zen nos dice que hagamos las cosas simples
  - Principio KISS : Keep it Short and Simple (Mantenlo corto y simple)
- Muchas veces el código es:
  - fácil de leer y entender.
  - fácil de escribir.

## Probar en el momento

Podemos pedirle a Geany que **ejecute una porción de código** para ver que hace.

¡Python es interpretado!

## Zen de Python

```
import this
```

- El Zen nos dice que hagamos las cosas simples
  - **Principio KISS** : Keep it Short and Simple (Mantenlo corto y simple)
- Muchas veces el código es:
  - fácil de leer y entender.
  - fácil de escribir.

## Probar en el momento

Podemos pedirle a Geany que **ejecute una porción de código** para ver que hace.

¡Python es interpretado!

## Zen de Python

```
import this
```

- El Zen nos dice que hagamos las cosas simples
  - **Principio KISS** : Keep it Short and Simple (Mantenlo corto y simple)
- Muchas veces el código es:
  - fácil de leer y entender.
  - fácil de escribir.

## Probar en el momento

Podemos pedirle a Geany que **ejecute una porción de código** para ver que hace.

¡Python es interpretado!

## Zen de Python

```
import this
```

- El Zen nos dice que hagamos las cosas simples
  - **Principio KISS** : Keep it Short and Simple (Mantenlo corto y simple)
- Muchas veces el código es:
  - fácil de leer y entender.
  - fácil de escribir.

## Probar en el momento

Podemos pedirle a Geany que **ejecute una porción de código** para ver que hace.

¡Python es interpretado!

## Zen de Python

```
import this
```

- El Zen nos dice que hagamos las cosas simples
  - **Principio KISS** : Keep it Short and Simple (Mantenlo corto y simple)
- Muchas veces el código es:
  - fácil de leer y entender.
  - fácil de escribir.

## 1 El Problema

- ¿Se Acuerdan?
- ¿Qué Otras Cosas Vimos?
- **Código Anterior**
- ¿Y cómo fue todo?

```
from tkinter import Tk
from tkinter.simpledialog import askstring as pedir_texto
from tkinter.messagebox import showinfo as mostrar_info
from tkinter.messagebox import askyesno as preg_sino

root = Tk()
root.withdraw()
```



```
cant_palmitos = 0

for i in range(20):
    nombre = pedir_texto('¡Pizzas!',
                          'Indique su nombre y apellido')
    plata = 500
    cant_personas = 20
    cada_uno = plata / cant_personas

    mostrar_info('¡Pizzas!', nombre + ' Te va a salir: '
                + str(cada_uno))
```

```
con_palmitos = preg_sino('¡Pizzas!', '¿Con palmitos?')

if con_palmitos:
    cant_palmitos = cant_palmitos + 1
    mostrar_info('¡Pizzas!', '¡Genial!')
else:
    cant_palmitos = cant_palmitos + 1
    mostrar_info('¡Pizzas!',
                '¡No te olvidés de avisarle a Juan!')

mostrar_info('¡Pizzas!', 'Juan, hay ' \
            + str(cant_palmitos)
            + ' pizzas con palmitos')
```

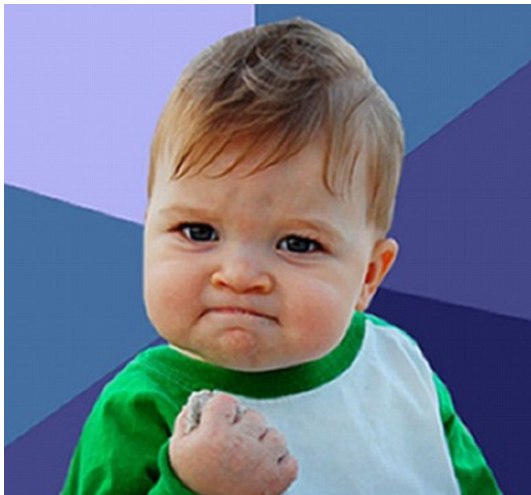
## 1 El Problema

- ¿Se Acuerdan?
- ¿Qué Otras Cosas Vimos?
- Código Anterior
- ¿Y cómo fue todo?

## ¿El resultado?

Juan hizo el software, dejó una compu en el curso, sus compañeros se inscribieron y así supo de antemano cuántos iban.

¡Fue todo un éxito!



# Outline

- 1 El Problema
- 2 Listas**
- 3 Tuplas
- 4 Hashes
- 5 Funciones
- 6 Muchas Gracias
- 7 Licencia



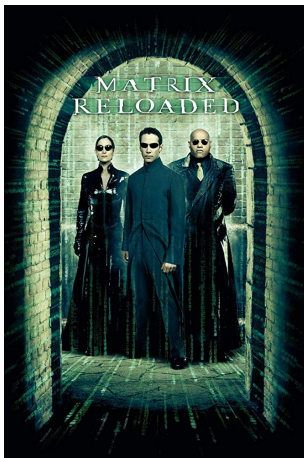
## 2 Listas

- Juan Quiere Saber Más...
- Crear la listas
- Agregar elementos
- Observaciones (Avanzado)
- Contar la cantidad
- Recorrer e imprimir
- Slices o cortes
- Código Completo
- Observaciones (Avanzado)
- ¿Qué más se puede hacer?

¡Oh no! ¡Otra vez!

Juan tuvo mucho éxito con su programa. Todo surgió tan bien, que le pidieron que organizara el próximo encuentro: la fiesta de fin de año del curso.

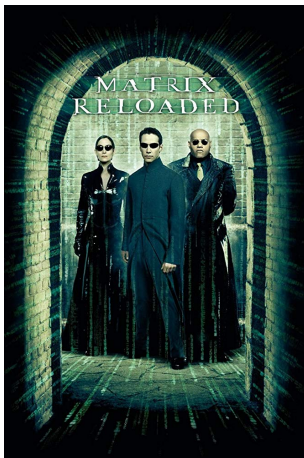
En otras palabras... Juan debe hacer un programa nuevo y mejor...



# ¡Oh no! ¡Otra vez!

Juan tuvo mucho éxito con su programa. Todo surgió tan bien, que le pidieron que organizara el próximo encuentro: la fiesta de fin de año del curso.

En otras palabras... Juan debe hacer un programa nuevo y mejor...





El problema es que ahora van a ir los que quieren y Juan debe hacer tarjetitas con el nombre de cada invitado. Para eso debe conocer los nombres de los que van a ir.

### ¿Qué debe hacer el programa?

- Armar una lista de los nombres vacía inicialmente.
- Si el estudiante quiere participar:
  - Pedirle que ingrese su nombre.
  - Guardarlo en la lista.
- Al terminar, imprimir los que se inscribieron.

### ¿Qué veremos?

- Crear listas
- Manipular listas
- Contar la cantidad de elementos en una lista
- Recorrer la lista

El problema es que ahora van a ir los que quieren y Juan debe hacer tarjetitas con el nombre de cada invitado. Para eso debe conocer los nombres de los que van a ir.

### ¿Qué debe hacer el programa?

- Armar una lista de los nombres vacía inicialmente.
- Si el estudiante quiere participar:
  - Permitir que ingrese su nombre.
  - guardarlo en la lista.
- Al terminar, imprimir los que se inscribieron.

### ¿Qué veremos?

- Crear listas
- Manipular listas
- Contar la cantidad de elementos en una lista
- Recorrer la lista

El problema es que ahora van a ir los que quieren y Juan debe hacer tarjetitas con el nombre de cada invitado. Para eso debe conocer los nombres de los que van a ir.

### ¿Qué debe hacer el programa?

- Armar una lista de los nombres vacía inicialmente.
- Si el estudiante quiere participar:
  - Permitir que ingrese su nombre.
  - guardarlo en la lista.
- Al terminar, imprimir los que se inscribieron.

### ¿Qué veremos?

- Crear listas
- Manipular listas
- Contar la cantidad de elementos en una lista
- Recorrer la lista

El problema es que ahora van a ir los que quieren y Juan debe hacer tarjetitas con el nombre de cada invitado. Para eso debe conocer los nombres de los que van a ir.

### ¿Qué debe hacer el programa?

- Armar una lista de los nombres vacía inicialmente.
- Si el estudiante quiere participar:
  - Permitir que ingrese su nombre.
  - guardarlo en la lista.
- Al terminar, imprimir los que se inscribieron.

### ¿Qué veremos?

- Crear listas
- Manipular listas
- Contar la cantidad de elementos en una lista
- Recorrer la lista

El problema es que ahora van a ir los que quieren y Juan debe hacer tarjetitas con el nombre de cada invitado. Para eso debe conocer los nombres de los que van a ir.

### ¿Qué debe hacer el programa?

- Armar una lista de los nombres vacía inicialmente.
- Si el estudiante quiere participar:
  - Permitir que ingrese su nombre.
  - guardarlo en la lista.
- Al terminar, imprimir los que se inscribieron.

### ¿Qué veremos?

- Crear listas
- Manipular listas
- Contar la cantidad de elementos en una lista
- Recorrer la lista

El problema es que ahora van a ir los que quieren y Juan debe hacer tarjetitas con el nombre de cada invitado. Para eso debe conocer los nombres de los que van a ir.

### ¿Qué debe hacer el programa?

- Armar una lista de los nombres vacía inicialmente.
- Si el estudiante quiere participar:
  - Permitir que ingrese su nombre.
  - guardarlo en la lista.
- Al terminar, imprimir los que se inscribieron.

### ¿Qué veremos?

- Crear listas
- Manipular listas
- Contar la cantidad de elementos en una lista
- Recorrer la lista

El problema es que ahora van a ir los que quieren y Juan debe hacer tarjetitas con el nombre de cada invitado. Para eso debe conocer los nombres de los que van a ir.

### ¿Qué debe hacer el programa?

- Armar una lista de los nombres vacía inicialmente.
- Si el estudiante quiere participar:
  - Permitir que ingrese su nombre.
  - guardarlo en la lista.
- Al terminar, imprimir los que se inscribieron.

### ¿Qué veremos?

- Crear listas
- Manipular listas
- Contar la cantidad de elementos en una lista
- Recorrer la lista

El problema es que ahora van a ir los que quieren y Juan debe hacer tarjetitas con el nombre de cada invitado. Para eso debe conocer los nombres de los que van a ir.

### ¿Qué debe hacer el programa?

- Armar una lista de los nombres vacía inicialmente.
- Si el estudiante quiere participar:
  - Permitir que ingrese su nombre.
  - guardarlo en la lista.
- Al terminar, imprimir los que se inscribieron.

### ¿Qué veremos?

- Crear listas
- Manipular listas
- Contar la cantidad de elementos en una lista
- Recorrer la lista



El problema es que ahora van a ir los que quieren y Juan debe hacer tarjetitas con el nombre de cada invitado. Para eso debe conocer los nombres de los que van a ir.

### ¿Qué debe hacer el programa?

- Armar una lista de los nombres vacía inicialmente.
- Si el estudiante quiere participar:
  - Permitir que ingrese su nombre.
  - guardarlo en la lista.
- Al terminar, imprimir los que se inscribieron.

### ¿Qué veremos?

- Crear listas
- Manipular listas
- Contar la cantidad de elementos en una lista
- Recorrer la lista

El problema es que ahora van a ir los que quieren y Juan debe hacer tarjetitas con el nombre de cada invitado. Para eso debe conocer los nombres de los que van a ir.

### ¿Qué debe hacer el programa?

- Armar una lista de los nombres vacía inicialmente.
- Si el estudiante quiere participar:
  - Permitir que ingrese su nombre.
  - guardarlo en la lista.
- Al terminar, imprimir los que se inscribieron.

### ¿Qué veremos?

- Crear listas
- Manipular listas
- Contar la cantidad de elementos en una lista
- Recorrer la lista

## 2 Listas

- Juan Quiere Saber Más...
- **Crear la listas**
- Agregar elementos
- Observaciones (Avanzado)
- Contar la cantidad
- Recorrer e imprimir
- Slices o cortes
- Código Completo
- Observaciones (Avanzado)
- ¿Qué más se puede hacer?

## ¿Qué son las Listas?

- Son contenedores de varias cosas
  - Puedo guardar muchos elementos adentro de una lista.
  - Elementos:
    - Textos (strings)
    - Números
    - Valores booleanos (True/False)

## Acceso

- Puedo acceder a esos elementos
  - “Dame el 5to elemento”
  - “Dame el n-ésimo elemento”

## ¿Qué son las Listas?

- Son contenedores de varias cosas
  - Puedo guardar muchos elementos adentro de una lista.
  - Elementos:
    - Textos (strings)
    - Números
    - Valores booleanos (True/False)

## Acceso

- Puedo acceder a esos elementos
  - “Dame el 5to elemento”
  - “Dame el n-ésimo elemento”

## ¿Qué son las Listas?

- Son contenedores de varias cosas
  - Puedo guardar muchos elementos adentro de una lista.
  - Elementos:
    - Textos (strings)
    - Números
    - Valores booleanos (True/False)

## Acceso

- Puedo acceder a esos elementos
  - “Dame el 5to elemento”
  - “Dame el n-ésimo elemento”

## ¿Qué son las Listas?

- Son contenedores de varias cosas
  - Puedo guardar muchos elementos adentro de una lista.
  - Elementos:
    - Textos (strings)
    - Números
    - Valores booleanos (True/False)

## Acceso

- Puedo acceder a esos elementos
  - “Dame el 5to elemento”
  - “Dame el n-ésimo elemento”

## ¿Qué son las Listas?

- Son contenedores de varias cosas
  - Puedo guardar muchos elementos adentro de una lista.
  - Elementos:
    - Textos (strings)
    - Números
    - Valores booleanos (True/False)

## Acceso

- Puedo acceder a esos elementos
  - “Dame el 5to elemento”
  - “Dame el n-ésimo elemento”



## ¿Qué son las Listas?

- Son contenedores de varias cosas
  - Puedo guardar muchos elementos adentro de una lista.
  - Elementos:
    - Textos (strings)
    - Números
    - Valores booleanos (True/False)

## Acceso

- Puedo acceder a esos elementos
  - “Dame el 5to elemento”
  - “Dame el n-ésimo elemento”

## Crear

```
lst_invitados = []  
# Seguro que van porque organizan el evento  
lst_invitados = ['Juan', 'Jorge', 'Cande']
```

## Acceder

Pedir un elemento de la lista es simple.

```
nombre = lst_invitados[0]
```

nombre tiene el primer elemento de la lista

- El índice es el número entre corchetes
- Se asigna el índice según el orden
- ¡El índice del primer elemento es 0!

## Crear

```
lst_invitados = []  
# Seguro que van porque organizan el evento  
lst_invitados = ['Juan', 'Jorge', 'Cande']
```

## Acceder

Pedir un elemento de la lista es simple.

```
nombre = lst_invitados[0]
```

nombre tiene el primer elemento de la lista

- El **índice** es el número entre corchetes
- Se asigna el índice según el orden
- ¡El índice del primer elemento es 0!

## Crear

```
lst_invitados = []  
# Seguro que van porque organizan el evento  
lst_invitados = ['Juan', 'Jorge', 'Cande']
```

## Acceder

Pedir un elemento de la lista es simple.

```
nombre = lst_invitados[0]
```

nombre tiene el primer elemento de la lista

- El **índice** es el número entre corchetes
- Se asigna el índice según el orden
- ¡El índice del primer elemento es 0!

## Crear

```
lst_invitados = []  
# Seguro que van porque organizan el evento  
lst_invitados = ['Juan', 'Jorge', 'Cande']
```

## Acceder

Pedir un elemento de la lista es simple.

```
nombre = lst_invitados[0]
```

nombre tiene el primer elemento de la lista

- El **índice** es el número entre corchetes
- Se asigna el índice según el orden
- ¡El índice del primer elemento es 0!

# ¡Podemos Usar Listas!

Para guardar los nombres: ¿se puede usar una lista!

¿Lo hacemos en el programa? ¿Dónde creamos nuestra lista de invitados?

Agregar al programa

```
# ...  
cant_palmitos = 0  
lst_invitados = [] # <- ¡nuevo!  
  
for i in range(20):  
# ...
```

Pero ¿y cómo la usamos?

Ya tenemos nuestra lista, pero...

¿Cómo agregamos los nombres?

# ¡Podemos Usar Listas!

Para guardar los nombres: ¿se puede usar una lista!

¿Lo hacemos en el programa? ¿Dónde creamos nuestra lista de invitados?

## Agregar al programa

```
# ...  
cant_palmitos = 0  
lst_invitados = [] # <- ¡nuevo!  
  
for i in range(20):  
# ...
```

Pero ¿y cómo la usamos?

Ya tenemos nuestra lista, pero...

¿Cómo agregamos los nombres?



# ¡Podemos Usar Listas!

Para guardar los nombres: ¿se puede usar una lista!

¿Lo hacemos en el programa? ¿Dónde creamos nuestra lista de invitados?

## Agregar al programa

```
# ...  
cant_palmitos = 0  
lst_invitados = [] # <- ¡nuevo!  
  
for i in range(20):  
# ...
```

## Pero ¿y cómo la usamos?

Ya tenemos nuestra lista, pero...

¿Cómo agregamos los nombres?





# ¡Podemos Usar Listas!

Para guardar los nombres: ¿se puede usar una lista!

¿Lo hacemos en el programa? ¿Dónde creamos nuestra lista de invitados?

## Agregar al programa

```
# ...
cant_palmitos = 0
lst_invitados = [] # <- ¡nuevo!

for i in range(20):
# ...
```

## Pero ¿y cómo la usamos?

Ya tenemos nuestra lista, pero...

¿Cómo agregamos los nombres?

# Para Probar (Avanzado)

## Para Probar (Avanzado)

- Mostrar con `tkinter` (`mostrar_info`) el elemento.
- ¿Qué pasa si usamos índices negativos?
- ¿Qué pasa si usamos `str()` en una lista?
  - `mostrar_info(str(lst_invitados[0]))`
  - `mostrar_info(str(lst_invitados))`
- ¿Qué pasa si creamos una lista con diferentes tipos de datos?

# Para Probar (Avanzado)

## Para Probar (Avanzado)

- Mostrar con `tkinter` (`mostrar_info`) el elemento.
- ¿Qué pasa si usamos índices negativos?
- ¿Qué pasa si usamos `str()` en una lista?
  - `mostrar_info(str(lst_invitados[0]))`
  - `mostrar_info(str(lst_invitados))`
- ¿Qué pasa si creamos una lista con diferentes tipos de datos?

# Para Probar (Avanzado)

## Para Probar (Avanzado)

- Mostrar con `tkinter` (`mostrar_info`) el elemento.
- ¿Qué pasa si usamos índices negativos?
- ¿Qué pasa si usamos `str()` en una lista?
  - `mostrar_info(str(lst_invitados[0]))`
  - `mostrar_info(str(lst_invitados))`
- ¿Qué pasa si creamos una lista con diferentes tipos de datos?

# Para Probar (Avanzado)

## Para Probar (Avanzado)

- Mostrar con `tkinter` (`mostrar_info`) el elemento.
- ¿Qué pasa si usamos índices negativos?
- ¿Qué pasa si usamos `str()` en una lista?
  - `mostrar_info(str(lst_invitados[0]))`
  - `mostrar_info(str(lst_invitados))`
- ¿Qué pasa si creamos una lista con diferentes tipos de datos?

# Para Probar (Avanzado)

## Para Probar (Avanzado)

- Mostrar con `tkinter` (`mostrar_info`) el elemento.
- ¿Qué pasa si usamos índices negativos?
- ¿Qué pasa si usamos `str()` en una lista?
  - `mostrar_info(str(lst_invitados[0]))`
  - `mostrar_info(str(lst_invitados))`
- ¿Qué pasa si creamos una lista con diferentes tipos de datos?

# Para Probar (Avanzado)

## Para Probar (Avanzado)

- Mostrar con `tkinter` (`mostrar_info`) el elemento.
- ¿Qué pasa si usamos índices negativos?
- ¿Qué pasa si usamos `str()` en una lista?
  - `mostrar_info(str(lst_invitados[0]))`
  - `mostrar_info(str(lst_invitados))`
- ¿Qué pasa si creamos una lista con diferentes tipos de datos?

## 2 Listas

- Juan Quiere Saber Más...
- Crear la listas
- **Agregar elementos**
- Observaciones (Avanzado)
- Contar la cantidad
- Recorrer e imprimir
- Slices o cortes
- Código Completo
- Observaciones (Avanzado)
- ¿Qué más se puede hacer?



A medida que van llegando estudiantes, sus nombres deben agregarse a la lista en orden.

- Primero llega Leandro, ingresa su nombre y se agrega a la lista.

- ['Leandro']

- Luego llega Claudia, ingresa su nombre y se agrega a la lista.

- ['Leandro', 'Claudia']

- ¡El orden es importante!

- ¿Agregamos al principio de la lista o al final?

- `lista.append('Leandro')`

- `lista.append('Claudia')`

A medida que van llegando estudiantes, sus nombres deben agregarse a la lista en orden.

- Primero llega Leandro, ingresa su nombre y se agrega a la lista.
  - ['Leandro']
- Luego llega Claudia, ingresa su nombre y se agrega a la lista.
  - ['Leandro', 'Claudia']
- ¡El orden es importante!
- ¿Agregamos al principio de la lista o al final?

```
l = ['Leandro', 'Claudia']
l.append('Pedro')
```
- ¿Agregamos al principio de la lista o al final?

```
l = ['Leandro', 'Claudia']
l.insert(0, 'Pedro')
```

A medida que van llegando estudiantes, sus nombres deben agregarse a la lista en orden.

- Primero llega Leandro, ingresa su nombre y se agrega a la lista.
  - ['Leandro']
- Luego llega Claudia, ingresa su nombre y se agrega a la lista.
  - ['Leandro', 'Claudia']
- ¡El orden es importante!
- ¿Agregamos al principio de la lista o al final?

A medida que van llegando estudiantes, sus nombres deben agregarse a la lista en orden.

- Primero llega Leandro, ingresa su nombre y se agrega a la lista.
  - ['Leandro']
- Luego llega Claudia, ingresa su nombre y se agrega a la lista.
  - ['Leandro', 'Claudia']

• ¡El orden es importante!

• ¿Agregamos al principio de la lista o al final?

A medida que van llegando estudiantes, sus nombres deben agregarse a la lista en orden.

- Primero llega Leandro, ingresa su nombre y se agrega a la lista.
  - ['Leandro']
- Luego llega Claudia, ingresa su nombre y se agrega a la lista.
  - ['Leandro', 'Claudia']

• ¡El orden es importante!

• ¿Agregamos al principio de la lista o al final?

A medida que van llegando estudiantes, sus nombres deben agregarse a la lista en orden.

- Primero llega Leandro, ingresa su nombre y se agrega a la lista.
  - ['Leandro']
- Luego llega Claudia, ingresa su nombre y se agrega a la lista.
  - ['Leandro', 'Claudia']
- ¡El orden es importante!
- ¿Agregamos al principio de la lista o al final?

Final `lst_invitados.append(nombre)`

Principio `lst_invitados.insert(0, nombre)`

A medida que van llegando estudiantes, sus nombres deben agregarse a la lista en orden.

- Primero llega Leandro, ingresa su nombre y se agrega a la lista.
  - ['Leandro']
- Luego llega Claudia, ingresa su nombre y se agrega a la lista.
  - ['Leandro', 'Claudia']
- ¡El orden es importante!
- ¿Agregamos al principio de la lista o al final?

Final `lst_invitados.append(nombre)`

Principio `lst_invitados.insert(0, nombre)`

A medida que van llegando estudiantes, sus nombres deben agregarse a la lista en orden.

- Primero llega Leandro, ingresa su nombre y se agrega a la lista.
  - ['Leandro']
- Luego llega Claudia, ingresa su nombre y se agrega a la lista.
  - ['Leandro', 'Claudia']
- ¡El orden es importante!
- ¿Agregamos al principio de la lista o al final?

**Final** `lst_invitados.append(nombre)`

**Principio** `lst_invitados.insert(0, nombre)`



A medida que van llegando estudiantes, sus nombres deben agregarse a la lista en orden.

- Primero llega Leandro, ingresa su nombre y se agrega a la lista.
  - ['Leandro']
- Luego llega Claudia, ingresa su nombre y se agrega a la lista.
  - ['Leandro', 'Claudia']
- ¡El orden es importante!
- ¿Agregamos al principio de la lista o al final?

**Final** `lst_invitados.append(nombre)`

**Principio** `lst_invitados.insert(0, nombre)`

¿Qué se debería modificar al programa para agregar los nombres en orden?

## Solución

```
# ...
nombre = pedir_texto('¡Pizzas!',
                    'Indique su nombre y apellido')

lst_invitados.append(nombre) # <- ¡nuevo!

plata = 500
# ...
```

¿Qué se debería modificar al programa para agregar los nombres en orden?

## Solución

```
# ...
nombre = pedir_texto('¡Pizzas!',
                    'Indique su nombre y apellido')

lst_invitados.append(nombre) # <- ¡nuevo!

plata = 500
# ...
```

## 2 Listas

- Juan Quiere Saber Más...
- Crear la listas
- Agregar elementos
- **Observaciones (Avanzado)**
- Contar la cantidad
- Recorrer e imprimir
- Slices o cortes
- Código Completo
- Observaciones (Avanzado)
- ¿Qué más se puede hacer?

- Una Lista es un Objeto
- Como todo objeto, recibe mensajes
- ¿Qué mensajes puede recibir?

En python `help(list)`

En Consola `pydoc list`

En Web `https:`

`//docs.python.org/3/library/stdtypes.html`

Concatenar, slices/corte, copiar, borrar, buscar min y max, buscar elementos, etc.

# Observaciones (Avanzado)

- Una Lista es un Objeto
- Como todo objeto, recibe mensajes
- ¿Qué mensajes puede recibir?

En python `help(list)`

En Consola `pydoc list`

En Web `https:`

`//docs.python.org/3/library/stdtypes.html`

Concatenar, slices/corte, copiar, borrar, buscar min y max, buscar elementos, etc.

# Observaciones (Avanzado)

- Una Lista es un Objeto
- Como todo objeto, recibe mensajes
- ¿Qué mensajes puede recibir?

En python `help(list)`

En Consola `pydoc list`

En Web `https:`

`//docs.python.org/3/library/stdtypes.html`

Concatenar, slices/corte, copiar, borrar, buscar min y max, buscar elementos, etc.

# Observaciones (Avanzado)

- Una Lista es un Objeto
- Como todo objeto, recibe mensajes
- ¿Qué mensajes puede recibir?

En python `help(list)`

En Consola `pydoc list`

En Web `https:`

`//docs.python.org/3/library/stdtypes.html`

Concatenar, slices/corte, copiar, borrar, buscar min y max, buscar elementos, etc.





# Observaciones (Avanzado)

- Una Lista es un Objeto
- Como todo objeto, recibe mensajes
- ¿Qué mensajes puede recibir?

En python `help(list)`

En Consola `pydoc list`

En Web [https:](https://docs.python.org/3/library/stdtypes.html)

[//docs.python.org/3/library/stdtypes.html](https://docs.python.org/3/library/stdtypes.html)  
Concatenar, slices/corte, copiar, borrar, buscar min y max, buscar elementos, etc.

# Observaciones (Avanzado)

- Una Lista es un Objeto
- Como todo objeto, recibe mensajes
- ¿Qué mensajes puede recibir?

En python `help(list)`

En Consola `pydoc list`

En Web `https:`

`//docs.python.org/3/library/stdtypes.html`

Concatenar, slices/corte, copiar, borrar, buscar min y max, buscar elementos, etc.

- Una Lista es un Objeto
- Como todo objeto, recibe mensajes
- ¿Qué mensajes puede recibir?

En python `help(list)`

En Consola `pydoc list`

En Web `https:`

`//docs.python.org/3/library/stdtypes.html`

Concatenar, slices/corte, copiar, borrar, buscar min y max, buscar elementos, etc.

## 2 Listas

- Juan Quiere Saber Más...
- Crear la listas
- Agregar elementos
- Observaciones (Avanzado)
- **Contar la cantidad**
- Recorrer e imprimir
- Slices o cortes
- Código Completo
- Observaciones (Avanzado)
- ¿Qué más se puede hacer?

# ¿Cuántos van a la fiesta?

Sería bueno que Juan pueda ver cuántos invitados hay inscriptos hasta el momento.

¿Podemos contar la cantidad de invitados?

## Contar los elementos

Para contar los elementos se usa `len`:

```
len(lst_invitados)
```

¿Se podría mostrar la cantidad de invitados a cada inscripción y al final?

## A cada inscripción

```
# ...
nombre = pedir_texto('¡Pizzas!',
                    'Indique su nombre y apellido')

lst_invitados.append(nombre)

cant_invitados = len(lst_invitados) # <- ¡nuevo!
mostrar_info(str(cant_invitados)) # <- ¡nuevo!

plata = 500
# ...
```

## Al final

```
# ...
    mostrar_info('¡Pizzas!', 'Juan, hay ' \
                + str(cant_palmitos)
                + ' pizzas con palmitos')

# ^^^ for termina aquí ^^^
cant_invitados = len(lst_invitados) # <- ¡nuevo!
mostrar_info('¡Pizzas!', 'Hay ' \
            + str(cant_invitados) \
            + ' inscriptos.') # <- ¡nuevo!
```

## 2 Listas

- Juan Quiere Saber Más...
- Crear la listas
- Agregar elementos
- Observaciones (Avanzado)
- Contar la cantidad
- Recorrer e imprimir
- Slices o cortes
- Código Completo
- Observaciones (Avanzado)
- ¿Qué más se puede hacer?



# Las Tarjetitas

Juan necesita los nombres para escribir en las tarjetas.

¿Cómo recorreremos la lista?

```
for nombre in lst_invitados:  
    mostrar_info('¡Pizzas!', nombre)
```

¿Mostramos todo junto?

```
todos = ""  
for nombre in lst_invitados:  
    todos = todos + ", " + nombre  
mostrar_info('¡Pizzas!', todos)
```

Juan necesita los nombres para escribir en las tarjetas.

¿Cómo recorreremos la lista?

```
for nombre in lst_invitados:  
    mostrar_info('¡Pizzas!', nombre)
```

¿Mostramos todo junto?

```
todos = ""  
for nombre in lst_invitados:  
    todos = todos + ", " + nombre  
mostrar_info('¡Pizzas!', todos)
```

Juan necesita los nombres para escribir en las tarjetas.

¿Cómo recorreremos la lista?

```
for nombre in lst_invitados:  
    mostrar_info('¡Pizzas!', nombre)
```

¿Mostramos todo junto?

```
todos = ""  
for nombre in lst_invitados:  
    todos = todos + ", " + nombre  
mostrar_info('¡Pizzas!', todos)
```

# Agregar al programa

Al terminar el programa, ¿los mostramos?

```
# ...
mostrar_info('¡Pizzas!', 'Juan, hay ' \
            + str(cant_palmitos)
            + ' pizzas con palmitos')

# ^^^ termina aquí ^^^
cant_invitados = len(lst_invitados)
mostrar_info('¡Pizzas!', 'Hay ' \
            + str(cant_invitados) \
            + ' inscriptos.')
```

```
# ^^^ lo agregamos antes ^^^
todos = "" # <- ¡nuevo!
for nombre in lst_invitados: # <- ¡nuevo!
    todos = todos + ", " + nombre # <- ¡nuevo!
mostrar_info('¡Pizzas!', todos) # <- ¡nuevo!
```



# Agregar al programa

Al terminar el programa, ¿los mostramos?

```
# ...
mostrar_info('¡Pizzas!', 'Juan, hay ' \
            + str(cant_palmitos)
            + ' pizzas con palmitos')

# ^^^ for termina aquí ^^^
cant_invitados = len(lst_invitados)
mostrar_info('¡Pizzas!', 'Hay ' \
            + str(cant_invitados) \
            + ' inscriptos.')
```

```
# ^^^ lo agregamos antes ^^^
todos = "" # <- ¡nuevo!
for nombre in lst_invitados: # <- ¡nuevo!
    todos = todos + ", " + nombre # <- ¡nuevo!
mostrar_info('¡Pizzas!', todos) # <- ¡nuevo!
```



# Agregar al programa

Al terminar el programa, ¿los mostramos?

```
# ...
mostrar_info('¡Pizzas!', 'Juan, hay ' \
            + str(cant_palmitos)
            + ' pizzas con palmitos')
# ^^^ for termina aquí ^^^
cant_invitados = len(lst_invitados)
mostrar_info('¡Pizzas!', 'Hay ' \
            + str(cant_invitados) \
            + ' inscriptos.')
```

```
# ^^^ lo agregamos antes ^^^
todos = "" # <- ¡nuevo!
for nombre in lst_invitados: # <- ¡nuevo!
    todos = todos + ", " + nombre # <- ¡nuevo!
mostrar_info('¡Pizzas!', todos) # <- ¡nuevo!
```



## 2 Listas

- Juan Quiere Saber Más...
- Crear la listas
- Agregar elementos
- Observaciones (Avanzado)
- Contar la cantidad
- Recorrer e imprimir
- Slices o cortes
- Código Completo
- Observaciones (Avanzado)
- ¿Qué más se puede hacer?

# Los 10 primeros tiene premio

Los profesores hablaron con Juan para convencerlo de que ofrezca un premio a los 10 primeros que se inscriban.

## ¿Qué debe hacer el programa?

- Mostrar los 10 primeros.

## ¿Qué veremos?

- Slices/cortes
- Listas comprensivas





¿Se puede hacer esto en Python?

## Slices

Cortes en listas se escriben así:

- `foo_list[desde:hasta]`
- `foo_list[desde:hasta:step]`

```
lst_inscriptos[0:10]
```



¿Se puede hacer esto en Python?

## Slices

Cortes en listas se escriben así:

- `foo_list[desde:hasta]`
- `foo_list[desde:hasta:step]`

```
lst_inscriptos[0:10]
```



¿Se puede hacer esto en Python?

## Slices

Cortes en listas se escriben así:

- `foo_list[desde:hasta]`
- `foo_list[desde:hasta:step]`

```
lst_inscriptos[0:10]
```



¿Se puede hacer esto en Python?

## Slices

Cortes en listas se escriben así:

- `foo_list[desde:hasta]`
- `foo_list[desde:hasta:step]`

```
lst_inscriptos[0:10]
```



¿Se puede hacer esto en Python?

## Slices

Cortes en listas se escriben así:

- `foo_list[desde:hasta]`
- `foo_list[desde:hasta:step]`

```
lst_inscriptos[0:10]
```

# Probar (Avanzado)

En Geany o en la consola de Python, probar lo siguiente.

## Probar

```
lst = list(range(10))  
lst[0:2]  
lst[0:8:2]
```

## Respuesta

- [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- [0, 1]
- [0, 2, 4, 6]

# Probar (Avanzado)

En Geany o en la consola de Python, probar lo siguiente.

## Probar

```
lst = list(range(10))  
lst[0:2]  
lst[0:8:2]
```

## Respuesta

- [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- [0, 1]
- [0, 2, 4, 6]

# Probar (Avanzado)

En Geany o en la consola de Python, probar lo siguiente.

## Probar

```
lst = list(range(10))  
lst[0:2]  
lst[0:8:2]
```

## Respuesta

- [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- [0, 1]
- [0, 2, 4, 6]



# Probar (Avanzado)

En Geany o en la consola de Python, probar lo siguiente.

## Probar

```
lst = list(range(10))  
lst[0:2]  
lst[0:8:2]
```

## Respuesta

- [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- [0, 1]
- [0, 2, 4, 6]

## ¿Y si usamos negativos? (Avanzado)

Los arreglos aceptan negativos como índices.

### Probar

```
lst[-7:-1]
```

```
lst[7:0:-2]
```

```
lst[-2:-8:-2]
```

### Respuesta

- [3, 4, 5, 6, 7, 8]
- [7, 5, 3, 1]
- [8, 6, 4]

## ¿Y si usamos negativos? (Avanzado)

Los arreglos aceptan negativos como índices.

### Probar

```
lst[-7:-1]
```

```
lst[7:0:-2]
```

```
lst[-2:-8:-2]
```

### Respuesta

- [3, 4, 5, 6, 7, 8]
- [7, 5, 3, 1]
- [8, 6, 4]

## ¿Y si usamos negativos? (Avanzado)

Los arreglos aceptan negativos como índices.

### Probar

```
lst[-7:-1]
```

```
lst[7:0:-2]
```

```
lst[-2:-8:-2]
```

### Respuesta

- [3, 4, 5, 6, 7, 8]
- [7, 5, 3, 1]
- [8, 6, 4]

## ¿Y si usamos negativos? (Avanzado)

Los arreglos aceptan negativos como índices.

### Probar

```
lst[-7:-1]
```

```
lst[7:0:-2]
```

```
lst[-2:-8:-2]
```

### Respuesta

- [3, 4, 5, 6, 7, 8]
- [7, 5, 3, 1]
- [8, 6, 4]

# Agregar al Programa

Al final, listemos los 10 primeros nombres para saber quienes ganaron el premio.

Agregar

```
premiados = ""
for nombre in lst_invitados[0:10]:
    premiados += ", " + nombre

mostrar_info(';Pizzas!', 'Los premiados son:' \
            + premiados)
```

# Agregar al Programa

Al final, listemos los 10 primeros nombres para saber quienes ganaron el premio.

## Agregar

```
premiados = ""
for nombre in lst_invitados[0:10]:
    premiados += ", " + nombre

mostrar_info(';Pizzas!', 'Los premiados son:' \
            + premiados)
```

## 2 Listas

- Juan Quiere Saber Más...
- Crear la listas
- Agregar elementos
- Observaciones (Avanzado)
- Contar la cantidad
- Recorrer e imprimir
- Slices o cortes
- **Código Completo**
- Observaciones (Avanzado)
- ¿Qué más se puede hacer?



# Primer Parte

```
from tkinter import Tk
from tkinter.simpledialog import askstring as pedir_texto
from tkinter.messagebox import showinfo as mostrar_info
from tkinter.messagebox import askyesno as preg_sino

root = Tk()
root.withdraw()

cant_palmitos = 0
lst_invitados = [] # <- ;nuevo!

for i in range(20):
    nombre = pedir_texto(';Pizzas!',
                        'Indique su nombre y apellido')

    lst_invitados.append(nombre) # <- ;nuevo!
    cant_invitados = len(lst_invitados) # <- ;nuevo!
    mostrar_info(str(cant_invitados)) # <- ;nuevo!
```



## Segunda Parte

```
plata = 500
cant_personas = 20
cada_uno = plata / cant_personas

mostrar_info('¡Pizzas!', nombre + ' Te va a salir: '
            + str(cada_uno))

con_palmitos = preg_sino('¡Pizzas!', '¿Con palmitos?')

if con_palmitos:
    cant_palmitos = cant_palmitos + 1
    mostrar_info('¡Pizzas!', '¡Genial!')
else:
    cant_palmitos = cant_palmitos + 1
    mostrar_info('¡Pizzas!',
                '¡No te olvidés de avisarle a Juan!')
```



```
mostrar_info('¡Pizzas!', 'Juan, hay ' \
            + str(cant_palmitos)
            + ' pizzas con palmitos')
```

```
cant_invitados = len(lst_invitados)    # <- ¡nuevo!
mostrar_info('¡Pizzas!', 'Hay ' \
            + str(cant_invitados) \
            + ' inscriptos.')
```

```
premiados = ""                        # <- ¡nuevo!
for nombre in lst_invitados[0:10]:    # <- ¡nuevo!
    premiados += ", " + nombre        # <- ¡nuevo!
```

```
mostrar_info('¡Pizzas!', 'Los premiados son:' \
            + premiados)                # <- ¡nuevo!
```

```
todos = ""                                # <- ¡nuevo!  
for nombre in lst_invitados:              # <- ¡nuevo!  
    todos = todos + ", " + nombre         # <- ¡nuevo!  
mostrar_info('¡Pizzas!', todos)          # <- ¡nuevo!
```

## 2 Listas

- Juan Quiere Saber Más...
- Crear la listas
- Agregar elementos
- Observaciones (Avanzado)
- Contar la cantidad
- Recorrer e imprimir
- Slices o cortes
- Código Completo
- **Observaciones (Avanzado)**
- ¿Qué más se puede hacer?

# Listas ¿Qué son? (Avanzado)

## Entonces... ¿Qué es una Lista?

- Objetos
- Contenedores
- Indexados con enteros
- Dinámicos
  - Sin tamaño límite
- Contenido:
  - cualquier dato
  - de variado tipo
  - puede repetir datos (no es un conjunto)

```
una_lista = [1, 2, 'Juan', 3.5, True, True, 2]
una_lista[3] == 'Juan'
mostrar_info(str(una_lista[2]))
```

# Listas ¿Qué son? (Avanzado)

## Entonces... ¿Qué es una Lista?

- Objetos
- Contenedores
- Indexados con enteros
- Dinámicos
  - Sin tamaño límite
- Contenido:
  - cualquier dato
  - de variado tipo
  - puede repetir datos (no es un conjunto)

```
una_lista = [1, 2, 'Juan', 3.5, True, True, 2]
una_lista[3] == 'Juan'
mostrar_info(str(una_lista[2]))
```

# Listas ¿Qué son? (Avanzado)

## Entonces... ¿Qué es una Lista?

- Objetos
- Contenedores
- Indexados con enteros
- Dimámicos
  - Sin tamaño límite
- Contenido:
  - cualquier dato
  - de variado tipo
  - puede repetir datos (no es un conjunto)

```
una_lista = [1, 2, 'Juan', 3.5, True, True, 2]
una_lista[3] == 'Juan'
mostrar_info(str(una_lista[2]))
```



## Entonces... ¿Qué es una Lista?

- Objetos
- Contenedores
- Indexados con enteros
- Dimámicos
  - Sin tamaño límite
- Contenido:
  - cualquier dato
  - de variado tipo
  - puede repetir datos (no es un conjunto)

```
una_lista = [1, 2, 'Juan', 3.5, True, True, 2]
una_lista[3] == 'Juan'
mostrar_info(str(una_lista[2]))
```

## Entonces... ¿Qué es una Lista?

- Objetos
- Contenedores
- Indexados con enteros
- Dinámicos
  - Sin tamaño límite
- Contenido:
  - cualquier dato
  - de variado tipo
  - puede repetir datos (no es un conjunto)

```
una_lista = [1, 2, 'Juan', 3.5, True, True, 2]
una_lista[3] == 'Juan'
mostrar_info(str(una_lista[2]))
```

## Entonces... ¿Qué es una Lista?

- Objetos
- Contenedores
- Indexados con enteros
- Dinámicos
  - Sin tamaño límite
- Contenido:
  - cualquier dato
  - de variado tipo
  - puede repetir datos (no es un conjunto)

```
una_lista = [1, 2, 'Juan', 3.5, True, True, 2]
una_lista[3] == 'Juan'
mostrar_info(str(una_lista[2]))
```

## Entonces... ¿Qué es una Lista?

- Objetos
- Contenedores
- Indexados con enteros
- Dimámicos
  - Sin tamaño límite
- Contenido:
  - cualquier dato
  - de variado tipo
  - puede repetir datos (no es un conjunto)

```
una_lista = [1, 2, 'Juan', 3.5, True, True, 2]
una_lista[3] == 'Juan'
mostrar_info(str(una_lista[2]))
```

## Entonces... ¿Qué es una Lista?

- Objetos
- Contenedores
- Indexados con enteros
- Dimámicos
  - Sin tamaño límite
- Contenido:
  - cualquier dato
  - de variado tipo
  - puede repetir datos (no es un conjunto)

```
una_lista = [1, 2, 'Juan', 3.5, True, True, 2]
una_lista[3] == 'Juan'
mostrar_info(str(una_lista[2]))
```

## Entonces... ¿Qué es una Lista?

- Objetos
- Contenedores
- Indexados con enteros
- Dinámicos
  - Sin tamaño límite
- Contenido:
  - cualquier dato
  - de variado tipo
  - puede repetir datos (no es un conjunto)

```
una_lista = [1, 2, 'Juan', 3.5, True, True, 2]
una_lista[3] == 'Juan'
mostrar_info(str(una_lista[2]))
```

## 2 Listas

- Juan Quiere Saber Más...
- Crear la listas
- Agregar elementos
- Observaciones (Avanzado)
- Contar la cantidad
- Recorrer e imprimir
- Slices o cortes
- Código Completo
- Observaciones (Avanzado)
- ¿Qué más se puede hacer?

- Crear una lista con distintos elementos (strings, enteros, reales, etc.).
- Armar la lista con nombre y apellido.
- Recorrer la lista para buscar a un elemento.
- Recorrer imprimiendo algo diferente por cada nombre.



- Crear una lista con distintos elementos (strings, enteros, reales, etc.).
- Armar la lista con nombre y apellido.
- Recorrer la lista para buscar a un elemento.
- Recorrer imprimiendo algo diferente por cada nombre.

- Crear una lista con distintos elementos (strings, enteros, reales, etc.).
- Armar la lista con nombre y apellido.
- Recorrer la lista para buscar a un elemento.
- Recorrer imprimiendo algo diferente por cada nombre.

- Crear una lista con distintos elementos (strings, enteros, reales, etc.).
- Armar la lista con nombre y apellido.
- Recorrer la lista para buscar a un elemento.
- Recorrer imprimiendo algo diferente por cada nombre.

# Outline

- 1 El Problema
- 2 Listas
- 3 Tuplas**
- 4 Hashes
- 5 Funciones
- 6 Muchas Gracias
- 7 Licencia



## 3 Tuplas

- Más información
- Código Completo
- Listas vs. Tuplas (Avanzado)
- ¿Qué más se puede hacer?

# ¿Pero van a querer con palmitos?

Cuando le dijeron a Juan que va a haber pizza, lo primero que se le ocurrió fue en preguntar si quieren con palmitos o no. Por experiencia, sabe que hay compañeros que no les gusta.

vs.



# ¿Con o sin Palmitos?

Sería ideal guardar en la lista el nombre y si quiere con palmitos o no.

## ¿Qué debe hacer el programa?

Lo mismo que antes pero... ..

- Guardar con el nombre si quiere con palmitos o no.

## ¿Qué veremos?

- Tuplas
- Diferencia entre Tuplas y Listas

# ¿Con o sin Palmitos?

Sería ideal guardar en la lista el nombre y si quiere con palmitos o no.

## ¿Qué debe hacer el programa?

Lo mismo que antes pero... ..

- Guardar con el nombre si quiere con palmitos o no.

## ¿Qué veremos?

- Tuplas
- Diferencia entre Tuplas y Listas



# ¿Con o sin Palmitos?

Sería ideal guardar en la lista el nombre y si quiere con palmitos o no.

## ¿Qué debe hacer el programa?

Lo mismo que antes pero... ..

- Guardar con el nombre si quiere con palmitos o no.

## ¿Qué veremos?

- Tuplas
- Diferencia entre Tuplas y Listas

# ¿Cómo sería nuestra lista?

## Cada elemento debe guardar

- El nombre.
- Si quiere palmitos o no.

Por ejemplo:

```
[  
    ('Jorge', True),  
    ('Cande', False),  
    ('Leandro', False),  
    ('Claudia', True),  
    ('Myriam', True)  
]
```

# ¿Cómo sería nuestra lista?

## Cada elemento debe guardar

- El nombre.
- Si quiere palmitos o no.

Por ejemplo:

```
[  
    ('Jorge', True),  
    ('Cande', False),  
    ('Leandro', False),  
    ('Claudia', True),  
    ('Myriam', True)  
]
```

# Crear Tuplas y Usarlas

Crear y usar las tuplas es como las listas.

```
dato = ('Jorge', True) # <- creamos  
mostrar_info(dato[0]) # <- accedemos
```

## Consideraciones

No se pueden cambiar sus elementos:

- No se pueden agregar más elementos.
- No se puede cambiar sus asignaciones.

# Crear Tuplas y Usarlas

Crear y usar las tuplas es como las listas.

```
dato = ('Jorge', True) # <- creamos
mostrar_info(dato[0]) # <- accedemos
```

## Consideraciones

No se pueden cambiar sus elementos:

- No se pueden agregar más elementos.
  - `dato.append(2)` **no funciona.**
- No se puede cambiar sus asignaciones.
  - `dato[0] = 'Leandro'` **no funciona.**

# Crear Tuplas y Usarlas

Crear y usar las tuplas es como las listas.

```
dato = ('Jorge', True) # <- creamos  
mostrar_info(dato[0]) # <- accedemos
```

## Consideraciones

No se pueden cambiar sus elementos:

- No se pueden agregar más elementos.
  - `dato.append(2)` **no funciona**.
- No se puede cambiar sus asignaciones.
  - `dato[0] = 'Leandro'` **no funciona**.

# Crear Tuplas y Usarlas

Crear y usar las tuplas es como las listas.

```
dato = ('Jorge', True) # <- creamos
mostrar_info(dato[0]) # <- accedemos
```

## Consideraciones

No se pueden cambiar sus elementos:

- No se pueden agregar más elementos.
  - `dato.append(2)` **no funciona.**
- No se puede cambiar sus asignaciones.
  - `dato[0] = 'Leandro'` **no funciona.**

# Crear Tuplas y Usarlas

Crear y usar las tuplas es como las listas.

```
dato = ('Jorge', True) # <- creamos  
mostrar_info(dato[0]) # <- accedemos
```

## Consideraciones

No se pueden cambiar sus elementos:

- No se pueden agregar más elementos.
  - `dato.append(2)` **no funciona.**
- No se puede cambiar sus asignaciones.
  - `dato[0] = 'Leandro'` **no funciona.**



# Agregar al Programa

¿Qué debería modificarse al programa?

```
# ...
```

```
    con_palmitos = preg_sino('¡Pizzas!', '¿Con palmitos?')
```

```
    lst_invitados.append( (nombre, con_palmitos) ) # <- ¡nuevo!
```

```
    if con_palmitos:
```

```
# ...
```

```
todos = ""
```

```
for dato in lst_invitados:                # <- ¡cambió!
```

```
    nombre = dato[0]                       # <- ¡nuevo!
```

```
    todos = todos + ", " + nombre
```

```
mostrar_info('¡Pizzas!', todos)
```



# Agregar al Programa

¿Qué debería modificarse al programa?

```
# ...
```

```
    con_palmitos = preg_sino('¡Pizzas!', '¿Con palmitos?')
```

```
    lst_invitados.append( (nombre, con_palmitos) ) # <- ¡nuevo!
```

```
    if con_palmitos:
```

```
# ...
```

```
todos = ""
```

```
for dato in lst_invitados:                # <- ¡cambió!
```

```
    nombre = dato[0]                       # <- ¡nuevo!
```

```
    todos = todos + ", " + nombre
```

```
mostrar_info('¡Pizzas!', todos)
```



# Agregar al Programa

¿Qué debería modificarse al programa?

```
# ...
```

```
    con_palmitos = preg_sino('¡Pizzas!', '¿Con palmitos?')
```

```
    lst_invitados.append( (nombre, con_palmitos) ) # <- ¡nuevo!
```

```
    if con_palmitos:
```

```
# ...
```

```
todos = ""
```

```
for dato in lst_invitados:                # <- ¡cambió!
```

```
    nombre = dato[0]                       # <- ¡nuevo!
```

```
    todos = todos + ", " + nombre
```

```
mostrar_info('¡Pizzas!', todos)
```



## 3 Tuplas

- Más información
- **Código Completo**
- Listas vs. Tuplas (Avanzado)
- ¿Qué más se puede hacer?

# Primera Parte

```
from tkinter import Tk
from tkinter.simpledialog import askstring as pedir_texto
from tkinter.messagebox import showinfo as mostrar_info
from tkinter.messagebox import askyesno as preg_sino

root = Tk()
root.withdraw()

cant_palmitos = 0
lst_invitados = [] # <- ¡nuevo!

for i in range(20):
    nombre = pedir_texto('¡Pizzas!',
                        'Indique su nombre y apellido')

    plata = 500
    cant_personas = 20
    cada_uno = plata / cant_personas
```



```
mostrar_info('¡Pizzas!', nombre + ' Te va a salir: '
            + str(cada_uno))

con_palmitos = preg_sino('¡Pizzas!', '¿Con palmitos?')

lst_invitados.append( (nombre, con_palmitos) ) # <- ¡nuevo!
cant_invitados = len(lst_invitados) # <- se movió
mostrar_info(str(cant_invitados)) # <- se movió

if con_palmitos:
    cant_palmitos = cant_palmitos + 1
    mostrar_info('¡Pizzas!', '¡Genial!')
else:
    cant_palmitos = cant_palmitos + 1
    mostrar_info('¡Pizzas!',
                '¡No te olvidés de avisarle a Juan!')
```

```
mostrar_info('¡Pizzas!', 'Juan, hay ' \
            + str(cant_palmitos)
            + ' pizzas con palmitos')
```

```
cant_invitados = len(lst_invitados)
mostrar_info('¡Pizzas!', 'Hay ' \
            + str(cant_invitados) \
            + ' inscriptos.')
```

```
premiados = ""
for dato in lst_invitados[0:10]:      # <- ¡cambió!
    nombre = dato[0]                  # <- ¡nuevo!
    premiados += ", " + nombre
```

```
mostrar_info('¡Pizzas!', 'Los premiados son:' \
            + premiados)              # <- ¡nuevo!
```

```
todos = ""
for dato in lst_invitados:
    nombre = dato[0]
    todos = todos + ", " + nombre
mostrar_info('¡Pizzas!', todos)

# <- ¡cambió!
# <- ¡nuevo!
```



## 3 Tuplas

- Más información
- Código Completo
- Listas vs. Tuplas (Avanzado)
- ¿Qué más se puede hacer?

## ¿Cuál es la diferencia? (Avanzado)

	Listas	Tuplas
<b>Contenedor</b>		
Tamaño	Dinámico	Fijo
Índice	0..Tamaño	0..Tamaño
<b>Contenido</b>		
Tipo Dato	Cualquiera	Cualquiera
Repite	Sí	Sí
Modificable	Sí	No

### Operadores

- Son mensajes.
- Ambos comparten los mismos mensajes.
- La implementación (métodos) son las mismas.

Ejemplo: + concatena.

(1, 2) + (3, 4) retorna (1, 2, 3, 4) y no (4, 6).

## ¿Cuál es la diferencia? (Avanzado)

	Listas	Tuplas
<b>Contenedor</b>		
Tamaño	Dinámico	Fijo
Índice	0..Tamaño	0..Tamaño
<b>Contenido</b>		
Tipo Dato	Cualquiera	Cualquiera
Repite	Sí	Sí
Modificable	Sí	No

### Operadores

- Son mensajes.
- Ambos comparten los mismos mensajes.
- La implementación (métodos) son las mismas.

Ejemplo: + concatena.

(1, 2) + (3, 4) retorna (1, 2, 3, 4) y no (4, 6).

## ¿Cuál es la diferencia? (Avanzado)

	Listas	Tuplas
<b>Contenedor</b>		
Tamaño	Dinámico	Fijo
Índice	0..Tamaño	0..Tamaño
<b>Contenido</b>		
Tipo Dato	Cualquiera	Cualquiera
Repite	Sí	Sí
Modificable	Sí	No

### Operadores

- Son mensajes.
- Ambos comparten los mismos mensajes.
- La implementación (métodos) son las mismas.

Ejemplo: + concatena.

(1, 2) + (3, 4) retorna (1, 2, 3, 4) y no (4, 6).

## ¿Cuál es la diferencia? (Avanzado)

	Listas	Tuplas
<b>Contenedor</b>		
Tamaño	Dinámico	Fijo
Índice	0..Tamaño	0..Tamaño
<b>Contenido</b>		
Tipo Dato	Cualquiera	Cualquiera
Repite	Sí	Sí
Modificable	Sí	No

### Operadores

- Son mensajes.
- Ambos comparten los mismos mensajes.
- La implementación (métodos) son las mismas.

Ejemplo: + concatena.

(1, 2) + (3, 4) retorna (1, 2, 3, 4) y no (4, 6).

## ¿Cuál es la diferencia? (Avanzado)

	Listas	Tuplas
<b>Contenedor</b>		
Tamaño	Dinámico	Fijo
Índice	0..Tamaño	0..Tamaño
<b>Contenido</b>		
Tipo Dato	Cualquiera	Cualquiera
Repite	Sí	Sí
Modificable	Sí	No

### Operadores

- Son mensajes.
- Ambos comparten los mismos mensajes.
- La implementación (métodos) son las mismas.

Ejemplo: + concatena.

(1, 2) + (3, 4) retorna (1, 2, 3, 4) y no (4, 6).

## 3 Tuplas

- Más información
- Código Completo
- Listas vs. Tuplas (Avanzado)
- ¿Qué más se puede hacer?

- **Agregar más datos (edad, cuantos invitados trae, etc.).**
- Recorrer la lista procesando las tuplas de diferente forma.
- Recorrer si le gusta con palmitos, imprimir "con palmitos".
- Recorrer y devolver una lista con solo las tuplas que les gusta con palmitos.
- Recorrer y separar en dos listas aquellos que les gusta y aquellos que no les gusta palmitos.



- Agregar más datos (edad, cuantos invitados trae, etc.).
- Recorrer la lista procesando las tuplas de diferente forma.
- Recorrer si le gusta con palmitos, imprimir "con palmitos".
- Recorrer y devolver una lista con solo las tuplas que les gusta con palmitos.
- Recorrer y separar en dos listas aquellos que les gusta y aquellos que no les gusta palmitos.

- Agregar más datos (edad, cuantos invitados trae, etc.).
- Recorrer la lista procesando las tuplas de diferente forma.
- Recorrer si le gusta con palmitos, imprimir "con palmitos".
- Recorrer y devolver una lista con solo las tuplas que les gusta con palmitos.
- Recorrer y separar en dos listas aquellos que les gusta y aquellos que no les gusta palmitos.

- Agregar más datos (edad, cuantos invitados trae, etc.).
- Recorrer la lista procesando las tuplas de diferente forma.
- Recorrer si le gusta con palmitos, imprimir "con palmitos".
- Recorrer y devolver una lista con solo las tuplas que les gusta con palmitos.
- Recorrer y separar en dos listas aquellos que les gusta y aquellos que no les gusta palmitos.

- Agregar más datos (edad, cuantos invitados trae, etc.).
- Recorrer la lista procesando las tuplas de diferente forma.
- Recorrer si le gusta con palmitos, imprimir "con palmitos".
- Recorrer y devolver una lista con solo las tuplas que les gusta con palmitos.
- Recorrer y separar en dos listas aquellos que les gusta y aquellos que no les gusta palmitos.

# Outline

- 1 El Problema
- 2 Listas
- 3 Tuplas
- 4 Hashes**
- 5 Funciones
- 6 Muchas Gracias
- 7 Licencia



## 4 Hashes

- ¿Qué ingredientes comprar?
- Código Completo

## ¿Qué ingredientes comprar?

Juan quiere saber qué ingredientes comprar para cada tipo de pizza. Por lo que debe armar un resumen de los tipos y la cantidad. Además, los que son premiados tienen una pizza con doble de palmitos (si les gusta) y doble queso.

### ¿Qué debe hacer el programa?

- Armar una lista de ingredientes.
- Recorrer los invitados:
  - Contar los ingredientes por cada invitado.
  - Contar los palmitos si al invitado le gusta.
  - Si el invitado está entre los 10 primeros:

### ¿Qué veremos?

- Hashes (diccionarios)

## ¿Qué ingredientes comprar?

Juan quiere saber qué ingredientes comprar para cada tipo de pizza. Por lo que debe armar un resumen de los tipos y la cantidad. Además, los que son premiados tienen una pizza con doble de palmitos (si les gusta) y doble queso.

### ¿Qué debe hacer el programa?

- Armar una lista de ingredientes.
- Recorrer los invitados:
  - Contar los ingredientes por cada invitado.
  - Contar los palmitos si al invitado le gusta.
  - Si el invitado está entre los 10 primeros:
    - Contar doble de palmitos si le gusta
    - Contar doble de queso

### ¿Qué veremos?

- Hashes (diccionarios)



## ¿Qué ingredientes comprar?

Juan quiere saber qué ingredientes comprar para cada tipo de pizza. Por lo que debe armar un resumen de los tipos y la cantidad. Además, los que son premiados tienen una pizza con doble de palmitos (si les gusta) y doble queso.

### ¿Qué debe hacer el programa?

- Armar una lista de ingredientes.
- Recorrer los invitados:
  - Contar los ingredientes por cada invitado.
  - Contar los palmitos si al invitado le gusta.
  - Si el invitado está entre los 10 primeros:
    - Contar doble de palmitos si le gusta
    - Contar doble de queso

### ¿Qué veremos?

- Hashes (diccionarios)

## ¿Qué ingredientes comprar?

Juan quiere saber qué ingredientes comprar para cada tipo de pizza. Por lo que debe armar un resumen de los tipos y la cantidad. Además, los que son premiados tienen una pizza con doble de palmitos (si les gusta) y doble queso.

### ¿Qué debe hacer el programa?

- Armar una lista de ingredientes.
- Recorrer los invitados:
  - Contar los ingredientes por cada invitado.
  - Contar los palmitos si al invitado le gusta.
  - Si el invitado está entre los 10 primeros:
    - Contar doble de palmitos si le gusta
    - Contar doble de queso

### ¿Qué veremos?

- Hashes (diccionarios)

## ¿Qué ingredientes comprar?

Juan quiere saber qué ingredientes comprar para cada tipo de pizza. Por lo que debe armar un resumen de los tipos y la cantidad. Además, los que son premiados tienen una pizza con doble de palmitos (si les gusta) y doble queso.

### ¿Qué debe hacer el programa?

- Armar una lista de ingredientes.
- Recorrer los invitados:
  - Contar los ingredientes por cada invitado.
  - Contar los palmitos si al invitado le gusta.
  - Si el invitado está entre los 10 primeros:
    - Contar doble de palmitos si le gusta
    - Contar doble de queso

### ¿Qué veremos?

- Hashes (diccionarios)

## ¿Qué ingredientes comprar?

Juan quiere saber qué ingredientes comprar para cada tipo de pizza. Por lo que debe armar un resumen de los tipos y la cantidad. Además, los que son premiados tienen una pizza con doble de palmitos (si les gusta) y doble queso.

### ¿Qué debe hacer el programa?

- Armar una lista de ingredientes.
- Recorrer los invitados:
  - Contar los ingredientes por cada invitado.
  - Contar los palmitos si al invitado le gusta.
  - Si el invitado está entre los 10 primeros:
    - Contar doble de palmitos si le gusta
    - Contar doble de queso

### ¿Qué veremos?

- Hashes (diccionarios)

## ¿Qué ingredientes comprar?

Juan quiere saber qué ingredientes comprar para cada tipo de pizza. Por lo que debe armar un resumen de los tipos y la cantidad. Además, los que son premiados tienen una pizza con doble de palmitos (si les gusta) y doble queso.

### ¿Qué debe hacer el programa?

- Armar una lista de ingredientes.
- Recorrer los invitados:
  - Contar los ingredientes por cada invitado.
  - Contar los palmitos si al invitado le gusta.
  - Si el invitado está entre los 10 primeros:
    - Contar doble de palmitos si le gusta
    - Contar doble de queso

### ¿Qué veremos?

- Hashes (diccionarios)

## ¿Qué ingredientes comprar?

Juan quiere saber qué ingredientes comprar para cada tipo de pizza. Por lo que debe armar un resumen de los tipos y la cantidad. Además, los que son premiados tienen una pizza con doble de palmitos (si les gusta) y doble queso.

### ¿Qué debe hacer el programa?

- Armar una lista de ingredientes.
- Recorrer los invitados:
  - Contar los ingredientes por cada invitado.
  - Contar los palmitos si al invitado le gusta.
  - Si el invitado está entre los 10 primeros:
    - Contar doble de palmitos si le gusta
    - Contar doble de queso

### ¿Qué veremos?

- Hashes (diccionarios)

## ¿Qué ingredientes comprar?

Juan quiere saber qué ingredientes comprar para cada tipo de pizza. Por lo que debe armar un resumen de los tipos y la cantidad. Además, los que son premiados tienen una pizza con doble de palmitos (si les gusta) y doble queso.

### ¿Qué debe hacer el programa?

- Armar una lista de ingredientes.
- Recorrer los invitados:
  - Contar los ingredientes por cada invitado.
  - Contar los palmitos si al invitado le gusta.
  - Si el invitado está entre los 10 primeros:
    - Contar doble de palmitos si le gusta
    - Contar doble de queso

### ¿Qué veremos?

- Hashes (diccionarios)

# ¿Una lista con ingredientes?

Podemos recorrer los invitados e ir incrementando dependiendo de los datos.

Nuestra pregunta es: ¿Cómo llevamos registro de los ingredientes?

- Una variable por ingrediente
  - No es muy eficiente (¿hay muchos ingredientes?)
- Una lista
  - ¿Cómo está?
- Una tupla
  - ¿Cómo está? (¿cómo se maneja el estado, cómo se maneja una lista?)
  - ¿Hay algún problema con los ingredientes no manejados?

Sería útil si los índices fueran los ingredientes.



# ¿Una lista con ingredientes?

Podemos recorrer los invitados e ir incrementando dependiendo de los datos.

Nuestra pregunta es: ¿Cómo llevamos registro de los ingredientes?

- Una variable por ingrediente
  - No muy dinámico: ¿si hay un nuevo ingrediente?
- Una lista
  - ¿Cómo sería?
- Una tupla
  - `ings = (cant_queso, cant_salsa, cant_palmitos)`
  - Un poco mejor, pero los ingredientes no son números

Sería útil si los índices fueran los ingredientes.

# ¿Una lista con ingredientes?

Podemos recorrer los invitados e ir incrementando dependiendo de los datos.

Nuestra pregunta es: ¿Cómo llevamos registro de los ingredientes?

- Una variable por ingrediente
  - No muy dinámico: ¿si hay un nuevo ingrediente?
- Una lista
  - ¿Cómo sería?
- Una tupla
  - `ings = (cant_queso, cant_salsa, cant_palmitos)`
  - Un poco mejor, pero los ingredientes no son números
  - `ings = ["queso", "salsa", "palmitos"]`

Sería útil si los índices fueran los ingredientes.

# ¿Una lista con ingredientes?

Podemos recorrer los invitados e ir incrementando dependiendo de los datos.

Nuestra pregunta es: ¿Cómo llevamos registro de los ingredientes?

- Una variable por ingrediente
  - No muy dinámico: ¿si hay un nuevo ingrediente?

- Una lista

- ¿Cómo sería?

- Una tupla

- `ings = (cant_queso, cant_salsa, cant_palmitos)`
  - Un poco mejor, pero los ingredientes no son números

`ings[0] = cant_queso`

Sería útil si los índices fueran los ingredientes.

# ¿Una lista con ingredientes?

Podemos recorrer los invitados e ir incrementando dependiendo de los datos.

Nuestra pregunta es: ¿Cómo llevamos registro de los ingredientes?

- Una variable por ingrediente
  - No muy dinámico: ¿si hay un nuevo ingrediente?
- Una lista
  - ¿Cómo sería?

## • Una tupla

- `ings = (cant_queso, cant_salsa, cant_palmitos)`
- Un poco mejor, pero los ingredientes no son números

`ings[0] = cant_queso`

Sería útil si los índices fueran los ingredientes.

# ¿Una lista con ingredientes?

Podemos recorrer los invitados e ir incrementando dependiendo de los datos.

Nuestra pregunta es: ¿Cómo llevamos registro de los ingredientes?

- Una variable por ingrediente
  - No muy dinámico: ¿si hay un nuevo ingrediente?
- Una lista
  - ¿Cómo sería?
- Una tupla
  - `ings = (cant_queso, cant_salsa, cant_palmitos)`
  - Un poco mejor, pero los ingredientes no son números
    - ¿ `ings[0]` es queso?

Sería útil si los índices fueran los ingredientes.

# ¿Una lista con ingredientes?

Podemos recorrer los invitados e ir incrementando dependiendo de los datos.

Nuestra pregunta es: ¿Cómo llevamos registro de los ingredientes?

- Una variable por ingrediente
  - No muy dinámico: ¿si hay un nuevo ingrediente?
- Una lista
  - ¿Cómo sería?
- Una tupla
  - `ings = (cant_queso, cant_salsa, cant_palmitos)`
  - Un poco mejor, pero los ingredientes no son números
    - ¿ `ings[0]` es queso?

Sería útil si los índices fueran los ingredientes.

# ¿Una lista con ingredientes?

Podemos recorrer los invitados e ir incrementando dependiendo de los datos.

Nuestra pregunta es: ¿Cómo llevamos registro de los ingredientes?

- Una variable por ingrediente
  - No muy dinámico: ¿si hay un nuevo ingrediente?
- Una lista
  - ¿Cómo sería?
- Una tupla
  - `ings = (cant_queso, cant_salsa, cant_palmitos)`
  - Un poco mejor, pero los ingredientes no son números
    - ¿ `ings[0]` es queso?

Sería útil si los índices fueran los ingredientes.

# ¿Una lista con ingredientes?

Podemos recorrer los invitados e ir incrementando dependiendo de los datos.

Nuestra pregunta es: ¿Cómo llevamos registro de los ingredientes?

- Una variable por ingrediente
  - No muy dinámico: ¿si hay un nuevo ingrediente?
- Una lista
  - ¿Cómo sería?
- Una tupla
  - `ings = (cant_queso, cant_salsa, cant_palmitos)`
  - Un poco mejor, pero los ingredientes no son números
    - ¿ `ings[0]` es queso?

Sería útil si los índices fueran los ingredientes.



# ¿Una lista con ingredientes?

Podemos recorrer los invitados e ir incrementando dependiendo de los datos.

Nuestra pregunta es: ¿Cómo llevamos registro de los ingredientes?

- Una variable por ingrediente
  - No muy dinámico: ¿si hay un nuevo ingrediente?
- Una lista
  - ¿Cómo sería?
- Una tupla
  - `ings = (cant_queso, cant_salsa, cant_palmitos)`
  - Un poco mejor, pero los ingredientes no son números
    - ¿ `ings[0]` es queso?

Sería útil si los índices fueran los ingredientes.

# ¿Una lista con ingredientes?

Podemos recorrer los invitados e ir incrementando dependiendo de los datos.

Nuestra pregunta es: ¿Cómo llevamos registro de los ingredientes?

- Una variable por ingrediente
  - No muy dinámico: ¿si hay un nuevo ingrediente?
- Una lista
  - ¿Cómo sería?
- Una tupla
  - `ings = (cant_queso, cant_salsa, cant_palmitos)`
  - Un poco mejor, pero los ingredientes no son números
    - ¿ `ings[0]` es queso?

Sería útil si los índices fueran los ingredientes.

## ¿Qué son?

- Los hashes son como las listas.
- Pero sus índices son strings

• ¡Se puede hacer `ingredientes['queso']` !

## ¿Cómo se crean?

```
ingredientes = { 'queso' : 0, 'palmitos': 0, 'salsa': 0 }
```

```
ingredientes = {}
```

```
ingredientes['queso'] = 0
```

```
ingredientes['palmitos'] = 0
```

```
ingredientes['salsa'] = 0
```

## ¿Qué son?

- Los hashes son como las listas.
- Pero sus índices son strings
  - ¡Se puede hacer `ingredientes['queso']` !

## ¿Cómo se crean?

```
ingredientes = { 'queso' : 0, 'palmitos': 0, 'salsa': 0 }
```

```
ingredientes = {}
```

```
ingredientes['queso'] = 0
```

```
ingredientes['palmitos'] = 0
```

```
ingredientes['salsa'] = 0
```

## ¿Qué son?

- Los hashes son como las listas.
- Pero sus índices son strings
  - ¡Se puede hacer `ingredientes['queso']` !

## ¿Cómo se crean?

```
ingredientes = { 'queso' : 0, 'palmitos': 0, 'salsa': 0 }
```

```
ingredientes = {}
```

```
ingredientes['queso'] = 0
```

```
ingredientes['palmitos'] = 0
```

```
ingredientes['salsa'] = 0
```

## ¿Qué son?

- Los hashes son como las listas.
- Pero sus índices son strings
  - ¡Se puede hacer `ingredientes['queso']` !

## ¿Cómo se crean?

```
ingredientes = { 'queso' : 0, 'palmitos': 0, 'salsa': 0 }
```

```
ingredientes = {}
```

```
ingredientes['queso'] = 0
```

```
ingredientes['palmitos'] = 0
```

```
ingredientes['salsa'] = 0
```

## ¿Qué son?

- Los hashes son como las listas.
- Pero sus índices son strings
  - ¡Se puede hacer `ingredientes['queso']` !

## ¿Cómo se crean?

```
ingredientes = { 'queso' : 0, 'palmitos': 0, 'salsa': 0 }
```

```
ingredientes = {}
```

```
ingredientes['queso'] = 0
```

```
ingredientes['palmitos'] = 0
```

```
ingredientes['salsa'] = 0
```

# Calcular ingredientes

- Calculamos los 10 primeros para después
- Recorrer cada invitado
- Incrementar queso y salsa
- Si le gusta palmitos:
  - Incrementar palmitos
- Si está en los 10 primeros:
  - Incrementar palmitos
  - Incrementar queso

```
premiados = lst_invitados[0:10]
for invitado in lst_invitados:
    ings['queso'] += 1
    ings['salsa'] += 1
    if invitado[1] == True:
        ings['palmitos'] += 1
    if invitado in premiados:
        ings['queso'] += 1
        if invitado[1] == True:
            ings['palmitos'] += 1
```



# Calcular ingredientes

- Calculamos los 10 primeros para después
- Recorrer cada invitado
- Incrementar queso y salsa
- Si le gusta palmitos:
  - Incrementar palmitos
- Si está en los 10 primeros:
  - Incrementar palmitos
  - Incrementar queso

```
premiados = lst_invitados[0:10]
for invitado in lst_invitados:
    ings['queso'] += 1
    ings['salsa'] += 1
    if invitado[1] == True:
        ings['palmitos'] += 1
    if invitado in premiados:
        ings['queso'] += 1
        if invitado[1] == True:
            ings['palmitos'] += 1
```



# Calcular ingredientes

- Calculamos los 10 primeros para después
- Recorrer cada invitado
- Incrementar queso y salsa
- Si le gusta palmitos:
  - Incrementar palmitos
- Si está en los 10 primeros:
  - Incrementar palmitos
  - Incrementar queso

```
premiados = lst_invitados[0:10]
for invitado in lst_invitados:
    ings['queso'] += 1
    ings['salsa'] += 1
    if invitado[1] == True:
        ings['palmitos'] += 1
    if invitado in premiados:
        ings['queso'] += 1
        if invitado[1] == True:
            ings['palmitos'] += 1
```



# Calcular ingredientes

- Calculamos los 10 primeros para después
- Recorrer cada invitado
- Incrementar queso y salsa
- Si le gusta palmitos:
  - Incrementar palmitos
- Si está en los 10 primeros:
  - Incrementar palmitos
  - Incrementar queso

```
premiados = lst_invitados[0:10]
for invitado in lst_invitados:
    ings['queso'] += 1
    ings['salsa'] += 1
    if invitado[1] == True:
        ings['palmitos'] += 1
    if invitado in premiados:
        ings['queso'] += 1
        if invitado[1] == True:
            ings['palmitos'] += 1
```

# Calcular ingredientes

- Calculamos los 10 primeros para después
- Recorrer cada invitado
- Incrementar queso y salsa
- Si le gusta palmitos:
  - Incrementar palmitos
- Si está en los 10 primeros:
  - Incrementar palmitos
  - Incrementar queso

```
premiados = lst_invitados[0:10]
for invitado in lst_invitados:
    ings['queso'] += 1
    ings['salsa'] += 1
    if invitado[1] == True:
        ings['palmitos'] += 1
    if invitado in premiados:
        ings['queso'] += 1
        if invitado[1] == True:
            ings['palmitos'] += 1
```

# Calcular ingredientes

- Calculamos los 10 primeros para después
- Recorrer cada invitado
- Incrementar queso y salsa
- Si le gusta palmitos:
  - Incrementar palmitos
- Si está en los 10 primeros:
  - Incrementar palmitos
  - Incrementar queso

```
premiados = lst_invitados[0:10]
for invitado in lst_invitados:
    ings['queso'] += 1
    ings['salsa'] += 1
    if invitado[1] == True:
        ings['palmitos'] += 1
    if invitado in premiados:
        ings['queso'] += 1
        if invitado[1] == True:
            ings['palmitos'] += 1
```



# Calcular ingredientes

- Calculamos los 10 primeros para después
- Recorrer cada invitado
- Incrementar queso y salsa
- Si le gusta palmitos:
  - Incrementar palmitos
- Si está en los 10 primeros:
  - Incrementar palmitos
  - Incrementar queso

```
premiados = lst_invitados[0:10]
for invitado in lst_invitados:
    ings['queso'] += 1
    ings['salsa'] += 1
    if invitado[1] == True:
        ings['palmitos'] += 1
    if invitado in premiados:
        ings['queso'] += 1
        if invitado[1] == True:
            ings['palmitos'] += 1
```



# Calcular ingredientes

- Calculamos los 10 primeros para después
- Recorrer cada invitado
- Incrementar queso y salsa
- Si le gusta palmitos:
  - Incrementar palmitos
- Si está en los 10 primeros:
  - Incrementar palmitos
  - Incrementar queso

```
premiados = lst_invitados[0:10]
for invitado in lst_invitados:
    ings['queso'] += 1
    ings['salsa'] += 1
    if invitado[1] == True:
        ings['palmitos'] += 1
    if invitado in premiados:
        ings['queso'] += 1
        if invitado[1] == True:
            ings['palmitos'] += 1
```



Al final del completo:

- El código que hicimos.
- Mostramos los resultados de cada ingrediente.

```
mostrar_info('Ingredientes', str(ings))
```



## 4 Hashes

- ¿Qué ingredientes comprar?
- Código Completo

# Primera Parte

```
from tkinter import Tk
from tkinter.simpledialog import askstring as pedir_texto
from tkinter.messagebox import showinfo as mostrar_info
from tkinter.messagebox import askyesno as preg_sino

root = Tk()
root.withdraw()

cant_palmitos = 0
lst_invitados = []

for i in range(20):
    nombre = pedir_texto(';Pizzas!',
                        'Indique su nombre y apellido')

    plata = 500
    cant_personas = 20
    cada_uno = plata / cant_personas
```



```
mostrar_info('¡Pizzas!', nombre + ' Te va a salir: '
            + str(cada_uno))

con_palmitos = preg_sino('¡Pizzas!', '¿Con palmitos?')

lst_invitados.append( (nombre, con_palmitos) )
cant_invitados = len(lst_invitados)
mostrar_info(str(cant_invitados))

if con_palmitos:
    cant_palmitos = cant_palmitos + 1
    mostrar_info('¡Pizzas!', '¡Genial!')
else:
    cant_palmitos = cant_palmitos + 1
    mostrar_info('¡Pizzas!',
                '¡No te olvides de avisarle a Juan!')
```

```
mostrar_info('¡Pizzas!', 'Juan, hay ' \
            + str(cant_palmitos)
            + ' pizzas con palmitos')
```

```
cant_invitados = len(lst_invitados)
mostrar_info('¡Pizzas!', 'Hay ' \
            + str(cant_invitados) \
            + ' inscriptos.')
```

```
premiados = ""
for dato in lst_invitados[0:10]:
    nombre = dato[0]
    premiados += ", " + nombre
```

```
mostrar_info('¡Pizzas!', 'Los premiados son:' \
            + premiados)
```

## Cuarta Parte

```
ings = {'queso':0, 'salsa': 0, 'palmitos': 0}
premiados = lst_invitados[0:10]      # <- ¡nuevo!
for invitado in lst_invitados:       # <- ¡nuevo!
    ings['queso'] += 1                 # <- ¡nuevo!
    ings['salsa'] += 1                 # <- ¡nuevo!
    if invitado[1] == True:           # <- ¡nuevo!
        ings['palmitos'] += 1         # <- ¡nuevo!
    if invitado in premiados:         # <- ¡nuevo!
        ings['queso'] += 1            # <- ¡nuevo!
        if invitado[1] == True:      # <- ¡nuevo!
            ings['palmitos'] += 1     # <- ¡nuevo!

todos = ""
for dato in lst_invitados:
    nombre = dato[0]
    todos = todos + ", " + nombre
mostrar_info('¡Pizzas!', todos)
```



# Outline

- 1 El Problema
- 2 Listas
- 3 Tuplas
- 4 Hashes
- 5 Funciones**
- 6 Muchas Gracias
- 7 Licencia



## 5 Funciones

- ¡Mucha Repetición! ¡Muy Complejo!
- Parámetros
- Código Completo
- Ante un Problema...

# ¡Mucha Repetición! ¡Muy Complejo!

Ahora que Juan programó para dos eventos, se dió cuenta de ciertas cosas:

- A veces, le piden cosas muy complejas.
  - ¿Se dieron cuenta que el código creció y se torna largo de leer?
  - ¿Qué pasaría si le mostramos el código a un compañero?
  - En un futuro... ¿Entenderíamos nuestro propio código?
- Muchas veces, le piden las mismas cosas para hacer.
  - ¿Se dieron cuenta que muchas cosas se hacen igual para ambas fiestas?

¿Podemos ayudar?

¿Podemos hacer algo para evitar esto?

- Muchas veces, debe copiar y pegar código porque se repite mucho.

¿Habrà alguna forma de evitar esto?



# ¡Mucha Repetición! ¡Muy Complejo!

Ahora que Juan programó para dos eventos, se dió cuenta de ciertas cosas:

- A veces, le piden cosas muy complejas.
  - ¿Se dieron cuenta que el código creció y se torna largo de leer?
  - ¿Qué pasaría si le mostramos el código a un compañero?
  - En un futuro... ¿Entenderíamos nuestro propio código?
- Muchas veces, le piden las mismas cosas para hacer.
  - ¿Se dieron cuenta que muchas cosas se hacen igual para ambas fiestas?

## ¿Qué podemos hacer?

¿Podemos reutilizar código?

- Muchas veces, debe copiar y pegar código porque se repite mucho.

¿Habrà alguna forma de evitar esto?

# ¡Mucha Repetición! ¡Muy Complejo!

Ahora que Juan programó para dos eventos, se dió cuenta de ciertas cosas:

- A veces, le piden cosas muy complejas.
  - ¿Se dieron cuenta que el código creció y se torna largo de leer?
  - ¿Qué pasaría si le mostramos el código a un compañero?
  - En un futuro... ¿Entenderíamos nuestro propio código?
- Muchas veces, le piden las mismas cosas para hacer.
  - ¿Se dieron cuenta que muchas cosas se hacen igual para ambas fiestas?

- Muchas veces, debe copiar y pegar código porque se repite mucho.

¿Habrá alguna forma de evitar esto?

# ¡Mucha Repetición! ¡Muy Complejo!

Ahora que Juan programó para dos eventos, se dió cuenta de ciertas cosas:

- A veces, le piden cosas muy complejas.
  - ¿Se dieron cuenta que el código creció y se torna largo de leer?
  - ¿Qué pasaría si le mostramos el código a un compañero?
  - En un futuro... ¿Entenderíamos nuestro propio código?
- Muchas veces, le piden las mismas cosas para hacer.
  - ¿Se dieron cuenta que muchas cosas se hacen igual para ambas fiestas?

- Muchas veces, debe copiar y pegar código porque se repite mucho.

¿Habrà alguna forma de evitar esto?

# ¡Mucha Repetición! ¡Muy Complejo!

Ahora que Juan programó para dos eventos, se dió cuenta de ciertas cosas:

- A veces, le piden cosas muy complejas.
  - ¿Se dieron cuenta que el código creció y se torna largo de leer?
  - ¿Qué pasaría si le mostramos el código a un compañero?
  - En un futuro... ¿Entenderíamos nuestro propio código?
- Muchas veces, le piden las mismas cosas para hacer.
  - ¿Se dieron cuenta que muchas cosas se hacen igual para ambas fiestas?
    - Pedir el nombre.
    - Preguntar si le gusta con palmitos.
- Muchas veces, debe copiar y pegar código porque se repite mucho.

¿Habrà alguna forma de evitar esto?

# ¡Mucha Repetición! ¡Muy Complejo!

Ahora que Juan programó para dos eventos, se dió cuenta de ciertas cosas:

- A veces, le piden cosas muy complejas.
  - ¿Se dieron cuenta que el código creció y se torna largo de leer?
  - ¿Qué pasaría si le mostramos el código a un compañero?
  - En un futuro... ¿Entenderíamos nuestro propio código?
- Muchas veces, le piden las mismas cosas para hacer.
  - ¿Se dieron cuenta que muchas cosas se hacen igual para ambas fiestas?
    - Pedir el nombre.
    - Preguntar si le gusta con palmitos.
- Muchas veces, debe copiar y pegar código porque se repite mucho.

¿Habrá alguna forma de evitar esto?

# ¡Mucha Repetición! ¡Muy Complejo!

Ahora que Juan programó para dos eventos, se dió cuenta de ciertas cosas:

- A veces, le piden cosas muy complejas.
  - ¿Se dieron cuenta que el código creció y se torna largo de leer?
  - ¿Qué pasaría si le mostramos el código a un compañero?
  - En un futuro... ¿Entenderíamos nuestro propio código?
- Muchas veces, le piden las mismas cosas para hacer.
  - ¿Se dieron cuenta que muchas cosas se hacen igual para ambas fiestas?
    - Pedir el nombre.
    - Preguntar si le gusta con palmitos.
- Muchas veces, debe copiar y pegar código porque se repite mucho.

¿Habrà alguna forma de evitar esto?

# ¡Mucha Repetición! ¡Muy Complejo!

Ahora que Juan programó para dos eventos, se dió cuenta de ciertas cosas:

- A veces, le piden cosas muy complejas.
  - ¿Se dieron cuenta que el código creció y se torna largo de leer?
  - ¿Qué pasaría si le mostramos el código a un compañero?
  - En un futuro... ¿Entenderíamos nuestro propio código?
- Muchas veces, le piden las mismas cosas para hacer.
  - ¿Se dieron cuenta que muchas cosas se hacen igual para ambas fiestas?
    - Pedir el nombre.
    - Preguntar si le gusta con palmitos.
- Muchas veces, debe copiar y pegar código porque se repite mucho.

¿Habrà alguna forma de evitar esto?

# ¡Mucha Repetición! ¡Muy Complejo!

Ahora que Juan programó para dos eventos, se dió cuenta de ciertas cosas:

- A veces, le piden cosas muy complejas.
  - ¿Se dieron cuenta que el código creció y se torna largo de leer?
  - ¿Qué pasaría si le mostramos el código a un compañero?
  - En un futuro... ¿Entenderíamos nuestro propio código?
- Muchas veces, le piden las mismas cosas para hacer.
  - ¿Se dieron cuenta que muchas cosas se hacen igual para ambas fiestas?
    - Pedir el nombre.
    - Preguntar si le gusta con palmitos.
- Muchas veces, debe copiar y pegar código porque se repite mucho.

¿Habrà alguna forma de evitar esto?



# ¡Mucha Repetición! ¡Muy Complejo!

Ahora que Juan programó para dos eventos, se dió cuenta de ciertas cosas:

- A veces, le piden cosas muy complejas.
  - ¿Se dieron cuenta que el código creció y se torna largo de leer?
  - ¿Qué pasaría si le mostramos el código a un compañero?
  - En un futuro... ¿Entenderíamos nuestro propio código?
- Muchas veces, le piden las mismas cosas para hacer.
  - ¿Se dieron cuenta que muchas cosas se hacen igual para ambas fiestas?
    - Pedir el nombre.
    - Preguntar si le gusta con palmitos.
- Muchas veces, debe copiar y pegar código porque se repite mucho.

¿Habrà alguna forma de evitar esto?

# ¡Mucha Repetición! ¡Muy Complejo!

Ahora que Juan programó para dos eventos, se dió cuenta de ciertas cosas:

- A veces, le piden cosas muy complejas.
  - ¿Se dieron cuenta que el código creció y se torna largo de leer?
  - ¿Qué pasaría si le mostramos el código a un compañero?
  - En un futuro... ¿Entenderíamos nuestro propio código?
- Muchas veces, le piden las mismas cosas para hacer.
  - ¿Se dieron cuenta que muchas cosas se hacen igual para ambas fiestas?
    - Pedir el nombre.
    - Preguntar si le gusta con palmitos.
- Muchas veces, debe copiar y pegar código porque se repite mucho.

¿Habrá alguna forma de evitar esto?

Por ejemplo, Juan tuvo que pedir el nombre y preguntar por palmitos en ambos eventos.

En vez de copiar y pegar, ¿se podría hacer algo?

## ¿Funciones?

- Si pudieramos ponerle un nombre a un pedazo de código...
- Y después llamamos a ese código por ese nombre...

Por ejemplo, Juan tuvo que pedir el nombre y preguntar por palmitos en ambos eventos.

En vez de copiar y pegar, ¿se podría hacer algo?

## ¿Funciones?

- Si pudieramos ponerle un nombre a un pedazo de código...
- Y después llamamos a ese código por ese nombre...

Por ejemplo, Juan tuvo que pedir el nombre y preguntar por palmitos en ambos eventos.

En vez de copiar y pegar, ¿se podría hacer algo?

## ¿Funciones?

- Si pudieramos ponerle un nombre a un pedazo de código...
- Y después llamamos a ese código por ese nombre...

## Pedir Nombre

En vez de escribir esto:

```
nombre = pedir_texto('¡Pizzas!',  
                    'Indique su nombre y apellido')
```

Escribir esto: `nombre = preg_nombre()`

## Calcular Promedio

En vez de escribir esto:

```
plata = 500  
cant_personas = 20  
cada_uno = plata / cant_personas
```

Escribir esto: `cada_uno = calc_promedio()`

# Sería interesante

## Pedir Nombre

En vez de escribir esto:

```
nombre = pedir_texto('¡Pizzas!',  
                    'Indique su nombre y apellido')
```

Escribir esto: `nombre = preg_nombre()`

## Calcular Promedio

En vez de escribir esto:

```
plata = 500  
cant_personas = 20  
cada_uno = plata / cant_personas
```

Escribir esto: `cada_uno = calc_promedio()`

## Pedir Nombre

En vez de escribir esto:

```
nombre = pedir_texto('¡Pizzas!',  
                    'Indique su nombre y apellido')
```

Escribir esto: `nombre = preg_nombre()`

## Calcular Promedio

En vez de escribir esto:

```
plata = 500  
cant_personas = 20  
cada_uno = plata / cant_personas
```

Escribir esto: `cada_uno = calc_promedio()`



## Pedir Nombre

En vez de escribir esto:

```
nombre = pedir_texto('¡Pizzas!',  
                    'Indique su nombre y apellido')
```

Escribir esto: `nombre = preg_nombre()`

## Calcular Promedio

En vez de escribir esto:

```
plata = 500  
cant_personas = 20  
cada_uno = plata / cant_personas
```

Escribir esto: `cada_uno = calc_promedio()`

# Funciones

Las funciones nos permiten hacer justamente esto:

- 1 Definimos una función una vez: Asociamos un nombre a un código.
- 2 Llamamos por ese nombre.

## Definición de una Función

¡Prestar atención al return!

```
def calc_promedio():  
    plata = 500  
    cant_personas = 20  
    promedio = plata / cant_personas  
    return promedio
```

## Llamada a la Función

¡Prestar atención a los paréntesis!

```
cada_uno = calc_promedio()
```

# Funciones

Las funciones nos permiten hacer justamente esto:

- 1 Definimos una función una vez: Asociamos un nombre a un código.
- 2 Llamamos por ese nombre.

## Definición de una Función

¡Prestar atención al return!

```
def calc_promedio():  
    plata = 500  
    cant_personas = 20  
    promedio = plata / cant_personas  
    return promedio
```

## Llamada a la Función

¡Prestar atención a los paréntesis!

```
cada_uno = calc_promedio()
```

# Funciones

Las funciones nos permiten hacer justamente esto:

- 1 Definimos una función una vez: Asociamos un nombre a un código.
- 2 Llamamos por ese nombre.

## Definición de una Función

¡Prestar atención al return!

```
def calc_promedio():  
    plata = 500  
    cant_personas = 20  
    promedio = plata / cant_personas  
    return promedio
```

## Llamada a la Función

¡Prestar atención a los paréntesis!

```
cada_uno = calc_promedio()
```

# Funciones

Las funciones nos permiten hacer justamente esto:

- 1 Definimos una función una vez: Asociamos un nombre a un código.
- 2 Llamamos por ese nombre.

## Definición de una Función

¡Prestar atención al return!

```
def calc_promedio():  
    plata = 500  
    cant_personas = 20  
    promedio = plata / cant_personas  
    return promedio
```

## Llamada a la Función

¡Prestar atención a los paréntesis!

```
cada_uno = calc_promedio()
```

# Funciones

Las funciones nos permiten hacer justamente esto:

- 1 Definimos una función una vez: Asociamos un nombre a un código.
- 2 Llamamos por ese nombre.

## Definición de una Función

¡Prestar atención al return!

```
def calc_promedio():  
    plata = 500  
    cant_personas = 20  
    promedio = plata / cant_personas  
    return promedio
```

## Llamada a la Función

¡Prestar atención a los paréntesis!

```
cada_uno = calc_promedio()
```

# ¿Cambiamos el código?

¿Se animan a cambiar el código usando funciones?

## Recordar

Lo ideal es poner las definiciones arriba del todo.

## Prueben con...

- Calcular Promedio.
- Preguntar nombre.
- Preguntar con palmitos.

Al principio, después de importar los módulos:

## Calcular Promedio

```
def calc_promedio():  
    plata = 500  
    cant_personas = 20  
    promedio = plata / cant_personas  
    return promedio
```

## Preguntar Nombre

```
def preg_nombre():  
    nombre = pedir_texto('¡Pizzas!',  
                          'Indique su nombre y apellido')  
    return nombre
```



Al principio, después de importar los módulos:

## Calcular Promedio

```
def calc_promedio():  
    plata = 500  
    cant_personas = 20  
    promedio = plata / cant_personas  
    return promedio
```

## Preguntar Nombre

```
def preg_nombre():  
    nombre = pedir_texto('¡Pizzas!',  
                          'Indique su nombre y apellido')  
    return nombre
```

## Preguntar con Palmitos

```
def preg_palmitos():  
    con_palmitos = preg_sino('¡Pizzas!', '¡Con palmitos?')  
    return con_palmitos
```

# Cambiar en el Programa

Usar las funciones donde corresponda.

```
# ...
nombre = preg_nombre()

cada_uno = calc_promedio()
# ...
con_palmitos = preg_palmitos()
#...
```

## 5 Funciones

- ¡Mucha Repetición! ¡Muy Complejo!
- **Parámetros**
- Código Completo
- Ante un Problema...

Algunas veces, para calcular algo requerimos de información que no está en la función.

Podemos entregar esta información extra por medio de parámetros.

¿Qué pasa si... ?

- Queremos calcular la cantidad de invitados y mostrarlo en pantalla.
  - ¡Necesitamos de la lista de invitados!
- Queremos obtener los nombres de todos los invitados y unirlos en un solo texto.
  - ¡Necesitamos de la lista de invitados!

Las funciones pueden definirse con parámetros.

¡No se puede!

```
def mostrar_cant_invitados():  
    cant_invitados = len(mis_invitados)  
    mostrar_info('¡Pizzas!', 'Hay ' \  
                + str(cant_invitados) \  
                + ' inscriptos.')
```

- ¿Dónde está `mis_invitados`?

- Recordar:

- Tratemos de no usar las variables globales dentro de una función
- Evitemos usar `lst_invitados`, `cant_palmitos`, etc. dentro de la fnc.

Las funciones pueden definirse con parámetros.

¡No se puede!

```
def mostrar_cant_invitados():  
    cant_invitados = len(mis_invitados)  
    mostrar_info('¡Pizzas!', 'Hay ' \  
                + str(cant_invitados) \  
                + ' inscriptos.')
```

- ¿Dónde está `mis_invitados`?
- Recordar:
  - Tratemos de no usar las variables globales dentro de una función
  - Evitemos usar `lst_invitados`, `cant_palmitos`, etc. dentro de la fnc.

Las funciones pueden definirse con parámetros.

¡No se puede!

```
def mostrar_cant_invitados():  
    cant_invitados = len(mis_invitados)  
    mostrar_info('¡Pizzas!', 'Hay ' \  
                + str(cant_invitados) \  
                + ' inscriptos.')
```

- ¿Dónde está `mis_invitados`?
- Recordar:
  - **Tratemos de no usar las variables globales dentro de una función**
  - Evitemos usar `lst_invitados`, `cant_palmitos`, etc. dentro de la fnc.



Las funciones pueden definirse con parámetros.

¡No se puede!

```
def mostrar_cant_invitados():  
    cant_invitados = len(mis_invitados)  
    mostrar_info('¡Pizzas!', 'Hay ' \  
                + str(cant_invitados) \  
                + ' inscriptos.')
```

- ¿Dónde está `mis_invitados`?
- Recordar:
  - **Tratemos de no usar las variables globales dentro de una función**
  - Evitemos usar `lst_invitados`, `cant_palmitos`, etc. dentro de la fnc.

## ¡No se puede!

```
def mostrar_cant_invitados():
    cant_invitados = len(mis_invitados)
    mostrar_info('¡Pizzas!', 'Hay ' \
                + str(cant_invitados) \
                + ' inscriptos.')
```

## Mejor

```
def mostrar_cant_invitados(mis_invitados):
    cant_invitados = len(mis_invitados)
    mostrar_info('¡Pizzas!', 'Hay ' \
                + str(cant_invitados) \
                + ' inscriptos.')
```

Para llamar a esta función debemos darle la lista de invitados:

```
mostrar_cant_invitados(lst_invitados)
```

# ¡Hagámoslo Mejor!

... pasado cierto tiempo ...

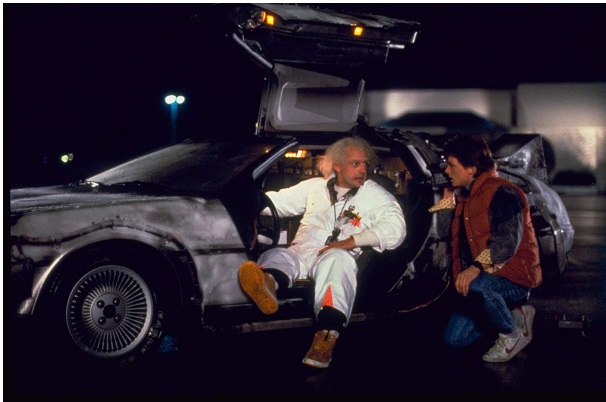
- Doc., doc., ¡¿Qué está pasando?!
- Martin, ¡No recuerdo para qué hice esta función!



# ¡Hagámoslo Mejor!

... pasado cierto tiempo ...

- Doc., doc., ¡¿Qué está pasando?!
- Martin, ¡No recuerdo para qué hice esta función!



# ¡Hagámoslo Mejor!

Python nos permite hacerlo mejor aún.

## ¿Qué hace la función?

```
def preg_nombre():  
    """  
    Pregunta el nombre al usuario.  
    """  
    nombre = pedir_texto('¡Pizzas!',  
                          'Indique su nombre y apellido')  
    return nombre
```

## ¿Qué devuelve la función?

```
def preg_palmitos() -> bool:
    """
    Pregunta al usuario si quiere con palmitos.
    """
    con_palmitos = preg_sino('¡Pizzas!', '¿Con palmitos?')
    return con_palmitos
```

## ¿Qué tipo son los parámetros?

```
def mostrar_cant_invitados(mis_invitados: list):  
    """  
    Mostrar la cantidad de invitados de la lista.  
    """  
  
    cant_invitados = len(mis_invitados)  
    mostrar_info('¡Pizzas!', 'Hay ' \  
                + str(cant_invitados) \  
                + ' inscriptos.')
```

## 5 Funciones

- ¡Mucha Repetición! ¡Muy Complejo!
- Parámetros
- Código Completo
- Ante un Problema...



# Primera Parte

```
from tkinter import Tk
from tkinter.simpledialog import askstring as pedir_texto
from tkinter.messagebox import showinfo as mostrar_info
from tkinter.messagebox import askyesno as preg_sino
```

```
def preg_nombre() -> str:
    """
    Pregunta el nombre al usuario.
    """
    nombre = pedir_texto('¡Pizzas!',
                        'Indique su nombre y apellido')
    return nombre
```

```
def preg_palmitos() -> bool:
    """
    Pregunta al usuario si quiere con palmitos.
    """
    con_palmitos = preg_sino('¡Pizzas!', '¡Con palmitos?')
    return con_palmitos
```



## Segunda Parte

```
def calc_promedio() -> float:
    """
    Calcula el promedio: lo que debe pagar cada uno de base.
    """
    plata = 500
    cant_personas = 20
    promedio = plata / cant_personas
    return promedio

def mostrar_cant_invitados(mis_invitados: list):
    """
    Mostrar la cantidad de invitados de la lista.
    """
    cant_invitados = len(mis_invitados)
    mostrar_info('¡Pizzas!', 'Hay ' \
                + str(cant_invitados) \
                + ' inscriptos.')
```



```
def juntar_nombres_invitados(mis_invitados: list) -> str:
    """
    Crear un string con todos los nombres de los invitados.
    """
    str_nombres = ""
    for dato in mis_invitados:
        nombre = dato[0]
        str_nombres = todos + ", " + nombre
    return str_nombres
```

```
root = Tk()
root.withdraw()

cant_palmitos = 0
lst_invitados = []

for i in range(20):
    nombre = preg_nombre() # <- ¡nuevo!

    cada_uno = calc_promedio() # <- ¡nuevo!
    mostrar_info('¡Pizzas!', nombre + ' Te va a salir: '
                + str(cada_uno))
```

```
con_palmitos = preg_palmitos() # <- ¡nuevo!  
  
lst_invitados.append( (nombre, con_palmitos)  
mostrar_cant_invitados(lst_invitados) # <- ¡nuevo!  
  
if con_palmitos:  
    cant_palmitos = cant_palmitos + 1  
    mostrar_info('¡Pizzas!', '¡Genial!')  
else:  
    cant_palmitos = cant_palmitos + 1  
    mostrar_info('¡Pizzas!',  
                '¡No te olvidés de avisarle a Juan!')
```

```
mostrar_info('¡Pizzas!', 'Juan, hay ' \  
            + str(cant_palmitos) \  
            + 'pizzas con palmitos')
```

```
mostrar_cant_invitados(lst_invitados) # <- ¡nuevo!
```

```
todos = juntar_nombres_invitados(lst_invitados) # <- ¡nuevo!  
mostrar_info('¡Pizzas!', todos)
```

## 5 Funciones

- ¡Mucha Repetición! ¡Muy Complejo!
- Parámetros
- Código Completo
- Ante un Problema...

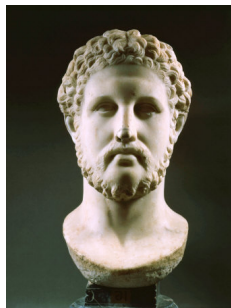
# Cuando es Muy Complejo



Ya lo decían los griegos y los romanos. . . *Dīvide et imperā*

Cuando nos enfrentamos a un problema podemos dividirlo en problemas más simples.





Ya lo decían los griegos y los romanos. . . *Dīvide et imperā*  
Cuando nos enfrentamos a un problema podemos dividirlo en problemas más simples.

Cuando nos enfrentamos a un problema podemos dividirlo en problemas más simples.

## Problema

Registrar los invitados.

¡Muy Genérico! ¡Muy Complejo!

## Registrar los invitados

- 1 Pedimos el nombre.
- 2 Preguntamos si quiere con palmitos.
- 3 Guardamos lo que el usuario ingresó.
- 4 Si se ingresaron 40 invitados termina, sino repite desde 1.

Cuando nos enfrentamos a un problema podemos dividirlo en problemas más simples.

## Problema

Registrar los invitados.

¡Muy Genérico! ¡Muy Complejo!

## Registrar los invitados

- Pedimos el nombre.
- Preguntamos si quiere con palmitos.
- Guardamos lo que el usuario ingresó.
- Si se ingresaron 40 invitados termina, sino repite desde 1.

Cuando nos enfrentamos a un problema podemos dividirlo en problemas más simples.

## Problema

Registrar los invitados.

¡Muy Genérico! ¡Muy Complejo!

## Registrar los invitados

- 1 Pedimos el nombre.
- 2 Preguntamos si quiere con palmitos.
- 3 Guardamos lo que el usuario ingresó.
- 4 Si se ingresaron 40 invitados termina, sino repite desde 1.

Cuando nos enfrentamos a un problema podemos dividirlo en problemas más simples.

## Problema

Registrar los invitados.

¡Muy Genérico! ¡Muy Complejo!

## Registrar los invitados

- 1 Pedimos el nombre.
- 2 Preguntamos si quiere con palmitos.
- 3 Guardamos lo que el usuario ingresó.
- 4 Si se ingresaron 40 invitados termina, sino repite desde 1.

Cuando nos enfrentamos a un problema podemos dividirlo en problemas más simples.

## Problema

Registrar los invitados.

¡Muy Genérico! ¡Muy Complejo!

## Registrar los invitados

- 1 Pedimos el nombre.
- 2 Preguntamos si quiere con palmitos.
- 3 Guardamos lo que el usuario ingresó.
- 4 Si se ingresaron 40 invitados termina, sino repite desde 1.

Cuando nos enfrentamos a un problema podemos dividirlo en problemas más simples.

## Problema

Registrar los invitados.

¡Muy Genérico! ¡Muy Complejo!

## Registrar los invitados

- 1 Pedimos el nombre.
- 2 Preguntamos si quiere con palmitos.
- 3 Guardamos lo que el usuario ingresó.
- 4 Si se ingresaron 40 invitados termina, sino repite desde 1.

# Enfrentándose a un Problema Complejo

Podemos encararlo de dos formas:

## Bottom-up

De abajo hacia arriba.

Vamos resolviendo los problemas simples que creamos necesarios. Luego, con estos resolvemos uno más complejo y así vamos construyendo hasta el que nos interese.

## Top-down

De arriba hacia abajo.

Tomamos el problema complejo y lo dividimos en subproblemas. Luego, esos subproblemas en otros y así hasta que sea abordable y fácil de resolver.



# Enfrentándose a un Problema Complejo

Podemos encararlo de dos formas:

## Bottom-up

De abajo hacia arriba.

Vamos resolviendo los problemas simples que creemos necesarios. Luego, con estos resolvemos uno más complejo y así vamos construyendo hasta el que nos interese.

## Top-down

De arriba hacia abajo.

Tomamos el problema complejo y lo dividimos en subproblemas. Luego, esos subproblemas en otros y así hasta que sea abordable y fácil de resolver.

# Enfrentándose a un Problema Complejo

Podemos encararlo de dos formas:

## Bottom-up

De abajo hacia arriba.

Vamos resolviendo los problemas simples que creemos necesarios. Luego, con estos resolvemos uno más complejo y así vamos construyendo hasta el que nos interese.

## Top-down

De arriba hacia abajo.

Tomamos el problema complejo y lo dividimos en subproblemas. Luego, esos subproblemas en otros y así hasta que sea abordable y fácil de resolver.

## Principio DRY

Do not Repeat Yourself! (¡No te repitas a tí mismo!)

## Consejos

- Cuando se puede modularizar, ¡hágalo!
- Cuando se repite código, ¡modularice!

## Principio DRY

Do not Repeat Yourself! (¡No te repitas a tí mismo!)

## Consejos

- Cuando se puede modularizar, ¡hágalo!
- Cuando se repite código, ¡modularice!
  - ¿Qué pasa si se repite código, pero una parte muy pequeña cambia?

## Principio DRY

Do not Repeat Yourself! (¡No te repitas a tí mismo!)

## Consejos

- Cuando se puede modularizar, ¡hágalo!
- Cuando se repite código, ¡modularice!
  - ¿Qué pasa si se repite código, pero una parte muy pequeña cambia?
    - ¡Usar parámetros!
    - Ej.: `def pedir_nombre(con_apellido: boolean):`

## Principio DRY

Do not Repeat Yourself! (¡No te repitas a tí mismo!)

## Consejos

- Cuando se puede modularizar, ¡hágalo!
- Cuando se repite código, ¡modularice!
  - ¿Qué pasa si se repite código, pero una parte muy pequeña cambia?
    - ¡Usar parámetros!
    - Ej.: `def pedir_nombre(con_apellido: boolean):`

## Principio DRY

Do not Repeat Yourself! (¡No te repitas a tí mismo!)

## Consejos

- Cuando se puede modularizar, ¡hágalo!
- Cuando se repite código, ¡modularice!
  - ¿Qué pasa si se repite código, pero una parte muy pequeña cambia?
    - ¡Usar parámetros!
    - Ej.: `def pedir_nombre(con_apellido: boolean):`

## Principio DRY

Do not Repeat Yourself! (¡No te repitas a tí mismo!)

## Consejos

- Cuando se puede modularizar, ¡hágalo!
- Cuando se repite código, ¡modularice!
  - ¿Qué pasa si se repite código, pero una parte muy pequeña cambia?
    - ¡Usar parámetros!
    - Ej.: `def pedir_nombre(con_apellido: boolean):`



# Outline

- 1 El Problema
- 2 Listas
- 3 Tuplas
- 4 Hashes
- 5 Funciones
- 6 Muchas Gracias**
- 7 Licencia

- 6 Muchas Gracias
  - Muchas Gracias

¡Muchas gracias!

# Outline

- 1 El Problema
- 2 Listas
- 3 Tuplas
- 4 Hashes
- 5 Funciones
- 6 Muchas Gracias
- 7 Licencia

- 7 Licencia
  - Licencia de Esta Obra
  - Imágenes

Excepto en los lugares que se ha indicado lo contrario:

Un Problema menos con Python se distribuye bajo una Licencia Creative Commons Atribución-SinDerivadas 4.0 Internacional.



## CC-By-ND

Excepto en los lugares que se ha indicado lo contrario:

Esta obra está licenciada bajo la Licencia Creative Commons Atribución-SinDerivadas 4.0 Internacional. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-nd/4.0/>.

- 7 Licencia
  - Licencia de Esta Obra
  - Imágenes



Obtenido

de

dailymail.co.uk.

Todos los derechos reservados.

<https://www.dailymail.co.uk/femail/article-3498922/>

The-face-launched-thousand-memes-Success-Kid-viral-star-baby-g  
html



Obtenido

de IMDB.com [https://www.](https://www.imdb.com/title/tt0234215/)

[imdb.com/title/tt0234215/](https://www.imdb.com/title/tt0234215/)

?ref\_=nv\_sr\_3?ref\_=nv\_sr\_3

Obtenido de fairwaypizza.com.

Todos los derechos reservados.

<https://fairwaypizza.com/pizza-and-wings/>



Obtenido

de

[www.quericavida.com](http://www.quericavida.com).





Todos los derechos reservados.

<https://www.quericavida.com/recipes/hearts-of-palm-pizza/496b251d-9086-409a-9e65-ec0823019937>



Obtenido

de

IMDB.com. Todos

los derechos reservados. [https://www.imdb.com/title/tt0088763/mediaindex?ref\\_=tt\\_pv\\_mi\\_sm](https://www.imdb.com/title/tt0088763/mediaindex?ref_=tt_pv_mi_sm)

Obtenido de Wikimedia

Commons. La imagen se

encuentra bajo la licencia Creative

Commons Attribution-Share

Alike 3.0 Unported. <https://commons.wikimedia.org/wiki/File:Philip-ii-of-macedon.jpg>



: float wrap: float wrap



Obtenido de [www.fool.com](http://www.fool.com). Todos los derechos reservados.

<https://www.fool.com/retirement/2019/06/07/sad-but-not-shocking-millennial-women-have-less-mo.aspx>





Obtenido  
desde [www.bhg.com.au](http://www.bhg.com.au).

Todos los derechos reservados.

<https://www.bhg.com.au/how-to-cut-bread>

