

Seguimos programando

ExactasPrograma

Facultad de Ciencias Exactas y Naturales, UBA

Verano 2023

RESUMEN Y UN POCO MÁS

```
dame_chance(resultados, cantidad_maxima)
```

- Implemente una función llamada `dame_chance` que tenga por parámetros una lista llamada `resultados` y el valor `cantidad_maxima` que indica la cantidad máxima de figus que podemos comprar.
- Debe devolver las *chances* de completar un álbum comprando como máximo `cantidad_maxima` de figus.
- Para esto, debemos revisar los elementos de la lista `resultados` y calcular el cociente entre la cantidad de veces que cada elemento es menor a `cantidad_maxima` (es decir, sirve para completar el álbum), dividido la cantidad total de elementos que hay en la lista `resultados`.

```
dame_chance(resultados, cantidad_maxima)
```

- Implemente una función llamada `dame_chance` que tenga por parámetros una lista llamada `resultados` y el valor `cantidad_maxima` que indica la cantidad máxima de figus que podemos comprar.
- Debe devolver las *chances* de completar un álbum comprando como máximo `cantidad_maxima` de figus.
- Para esto, debemos revisar los elementos de la lista `resultados` y calcular el cociente entre la cantidad de veces que cada elemento es menor a `cantidad_maxima` (es decir, sirve para completar el álbum), dividido la cantidad total de elementos que hay en la lista `resultados`.

Antes de seguir... ¿escribimos algo entre todos en el pizarrón?

`dame_chance(resultados, cantidad_maxima)`

- Implemente una función llamada `dame_chance` que tenga por parámetros una lista llamada `resultados` y el valor `cantidad_maxima` que indica la cantidad máxima de figus que podemos comprar.
- Debe devolver las *chances* de completar un álbum comprando como máximo `cantidad_maxima` de figus.
- Para esto, debemos revisar los elementos de la lista `resultados` y calcular el cociente entre la cantidad de veces que cada elemento es menor a `cantidad_maxima` (es decir, sirve para completar el álbum), dividido la cantidad total de elementos que hay en la lista `resultados`.

Antes de seguir... ¿escribimos algo entre todos en el pizarrón?

```
def dame_chance(resultados, cantidad_maxima):
    cuantos = 0
    i = 0
    while i < len(resultados):
        if resultados[i] <= cantidad_maxima:
            cuantos = cuantos + 1
        i = i + 1
    salida = cuantos / len(resultados)
    return salida
```

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - $<$ menor

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - $<$ menor
 - \leq menor o igual

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - $<$ menor
 - \leq menor o igual
 - $>$ mayor

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - $<$ menor
 - \leq menor o igual
 - $>$ mayor
 - \geq mayor o igual

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - $<$ menor
 - \leq menor o igual
 - $>$ mayor
 - \geq mayor o igual
 - $==$ igual

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - $<$ menor
 - $<=$ menor o igual
 - $>$ mayor
 - $>=$ mayor o igual
 - $==$ igual
 - $!=$ distinto

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - $<$ menor
 - $<=$ menor o igual
 - $>$ mayor
 - $>=$ mayor o igual
 - $==$ igual
 - $!=$ distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.
 - **and** se usa `x and y`. Solo da `True` cuando `x` e `y` son `True`.

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.
 - **and** se usa `x and y`. Solo da `True` cuando `x` e `y` son `True`.
 - **or** se usa `x or y`. Da `True` cuando alguna de las dos (o las dos) es `True`.

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.
 - **and** se usa `x and y`. Solo da `True` cuando `x` e `y` son `True`.
 - **or** se usa `x or y`. Da `True` cuando alguna de las dos (o las dos) es `True`.
- Esto aplica tanto para las condiciones del `if` como a las del `while`.

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.
 - **and** se usa `x and y`. Solo da `True` cuando `x` e `y` son `True`.
 - **or** se usa `x or y`. Da `True` cuando alguna de las dos (o las dos) es `True`.
- Esto aplica tanto para las condiciones del `if` como a las del `while`.

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.
 - **and** se usa `x and y`. Solo da `True` cuando `x` e `y` son `True`.
 - **or** se usa `x or y`. Da `True` cuando alguna de las dos (o las dos) es `True`.
- Esto aplica tanto para las condiciones del `if` como a las del `while`.

```
a = 7
x = 5
y = 9
res = a > x and a < y
print(res)
```

¿Qué valor se imprime?

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.
 - **and** se usa `x and y`. Solo da `True` cuando `x` e `y` son `True`.
 - **or** se usa `x or y`. Da `True` cuando alguna de las dos (o las dos) es `True`.
- Esto aplica tanto para las condiciones del `if` como a las del `while`.

```
a = 7
x = 5
y = 9
res = a > x and a < y
print(res)
```

¿Qué valor se imprime?

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.
 - **and** se usa `x and y`. Solo da `True` cuando `x` e `y` son `True`.
 - **or** se usa `x or y`. Da `True` cuando alguna de las dos (o las dos) es `True`.
- Esto aplica tanto para las condiciones del `if` como a las del `while`.

```
a = 7
x = 5
y = 9
res = a > x and a < y
print(res)
```

¿Qué valor se imprime? `True`

```
a = 2
x = 5
y = 0
res = a > x or a < y
print(res)
```

¿Qué valor se imprime?

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.
 - **and** se usa `x and y`. Solo da `True` cuando `x` e `y` son `True`.
 - **or** se usa `x or y`. Da `True` cuando alguna de las dos (o las dos) es `True`.
- Esto aplica tanto para las condiciones del `if` como a las del `while`.

```
a = 7
x = 5
y = 9
res = a>x and a<y
print(res)
```

¿Qué valor se imprime? `True`

```
a = 2
x = 5
y = 0
res = a>x or a<y
print(res)
```

¿Qué valor se imprime?

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.
 - **and** se usa `x and y`. Solo da `True` cuando `x` e `y` son `True`.
 - **or** se usa `x or y`. Da `True` cuando alguna de las dos (o las dos) es `True`.
- Esto aplica tanto para las condiciones del `if` como a las del `while`.

```
a = 7
x = 5
y = 9
res = a > x and a < y
print(res)
```

¿Qué valor se imprime? `True`

```
a = 2
x = 5
y = 0
res = a > x or a < y
print(res)
```

¿Qué valor se imprime? `False`

ACTIVIDAD DE HOY

Diez mil: ¿lo conoces?

Una excusa para romper el hielo



- ¿Cómo se juega?

Diez mil: ¿lo conoces?

Una excusa para romper el hielo



- ¿Cómo se juega?
- ¿Jugaste? ¿Dónde? ¿Cuándo?

Diez mil: ¿lo conoces?

Una excusa para romper el hielo



- ¿Cómo se juega?
- ¿Jugaste? ¿Dónde? ¿Cuándo?
- ¿Ganaste?

Diez mil:



- Es uno juego popular de dados, ideal para días de lluvia.

Diez mil:



- Es uno juego popular de dados, ideal para días de lluvia.
- Se juega entre varios jugadores (más de dos).

Diez mil:



- Es uno juego popular de dados, ideal para días de lluvia.
- Se juega entre varios jugadores (más de dos).
- Se usan cinco dados.

Diez mil:



- Es uno juego popular de dados, ideal para días de lluvia.
- Se juega entre varios jugadores (más de dos).
- Se usan cinco dados.
- Cada jugador va tirando los dados y sumando el puntaje que estos indican.

Diez mil:



- Es uno juego popular de dados, ideal para días de lluvia.
- Se juega entre varios jugadores (más de dos).
- Se usan cinco dados.
- Cada jugador va tirando los dados y sumando el puntaje que estos indican.
- El objetivo es llegar a 10.000 puntos.

Diez mil:



- Es uno juego popular de dados, ideal para días de lluvia.
- Se juega entre varios jugadores (más de dos).
- Se usan cinco dados.
- Cada jugador va tirando los dados y sumando el puntaje que estos indican.
- El objetivo es llegar a 10.000 puntos.

Diez mil: versión “simplificada” (por no decir otro juego...)

Vamos a tomar el *espíritu* del 10.000 para armar un juego que podamos implementar con un programa, así que las reglas van a estar relajadas (ya las vamos a ir viendo).

Puntajes - algunos ejemplos

- por cada 1 obtenido, se suman 100 puntos, mientras que 3 unos dan 1000 puntos. 4 unos suman 1100 puntos mientras que 5 unos dan 10000 puntos.
- por cada 5 obtenido, se suman 50 puntos, mientras que 3 cincos dan 500 puntos. 4 cincos suman 550 puntos mientras que 5 cincos dan 600 puntos.
- 2,3,4 y 6 no dan puntos.

Puntajes - algunos ejemplos

- por cada 1 obtenido, se suman 100 puntos, mientras que 3 unos dan 1000 puntos. 4 unos suman 1100 puntos mientras que 5 unos dan 10000 puntos.
- por cada 5 obtenido, se suman 50 puntos, mientras que 3 cincos dan 500 puntos. 4 cincos suman 550 puntos mientras que 5 cincos dan 600 puntos.
- 2,3,4 y 6 no dan puntos.



Puntajes - algunos ejemplos

- por cada 1 obtenido, se suman 100 puntos, mientras que 3 unos dan 1000 puntos. 4 unos suman 1100 puntos mientras que 5 unos dan 10000 puntos.
- por cada 5 obtenido, se suman 50 puntos, mientras que 3 cincos dan 500 puntos. 4 cincos suman 550 puntos mientras que 5 cincos dan 600 puntos.
- 2,3,4 y 6 no dan puntos.



Puntajes - algunos ejemplos

- por cada 1 obtenido, se suman 100 puntos, mientras que 3 unos dan 1000 puntos. 4 unos suman 1100 puntos mientras que 5 unos dan 10000 puntos.
- por cada 5 obtenido, se suman 50 puntos, mientras que 3 cincos dan 500 puntos. 4 cincos suman 550 puntos mientras que 5 cincos dan 600 puntos.
- 2,3,4 y 6 no dan puntos.



Puntajes - algunos ejemplos

- por cada 1 obtenido, se suman 100 puntos, mientras que 3 unos dan 1000 puntos. 4 unos suman 1100 puntos mientras que 5 unos dan 10000 puntos.
- por cada 5 obtenido, se suman 50 puntos, mientras que 3 cincos dan 500 puntos. 4 cincos suman 550 puntos mientras que 5 cincos dan 600 puntos.
- 2,3,4 y 6 no dan puntos.



Puntajes - algunos ejemplos

- por cada 1 obtenido, se suman 100 puntos, mientras que 3 unos dan 1000 puntos. 4 unos suman 1100 puntos mientras que 5 unos dan 10000 puntos.
- por cada 5 obtenido, se suman 50 puntos, mientras que 3 cincos dan 500 puntos. 4 cincos suman 550 puntos mientras que 5 cincos dan 600 puntos.
- 2,3,4 y 6 no dan puntos.



Puntajes - algunos ejemplos

- por cada 1 obtenido, se suman 100 puntos, mientras que 3 unos dan 1000 puntos. 4 unos suman 1100 puntos mientras que 5 unos dan 10000 puntos.
- por cada 5 obtenido, se suman 50 puntos, mientras que 3 cincos dan 500 puntos. 4 cincos suman 550 puntos mientras que 5 cincos dan 600 puntos.
- 2,3,4 y 6 no dan puntos.



¿Cómo se juega?

- Cada participante comienza con 0 puntos.
- Cada participante, en cada **jugada**, lanza cinco dados.
- Se calculan los puntajes según las siguientes reglas:
 - por cada 1 obtenido, se suman 100 puntos, mientras que 3 unos dan 1000 puntos. 4 unos suman 1100 puntos mientras que 5 unos dan 10000 puntos.
 - por cada 5 obtenido, se suman 50 puntos, mientras que 3 cincos dan 500 puntos. 4 cincos suman 550 puntos mientras que 5 cincos dan 600 puntos.
 - 2,3,4 y 6 no dan puntos.
- En cada ronda cada participante realiza una **jugada**,
- Al finalizar la ronda se suman los puntos obtenidos por cada jugador a los que ya tenía.
- El juego termina cuando al cabo de una ronda algún jugador alcanza (o supera) los 10.000 puntos.
- Vamos a utilizar k para denotar la cantidad de jugadores.

A JUGAR!

5 rondas

- Hacer grupos con $k = 4$ jugadores
- Jugar al 1000 (en lugar de 10mil)
- Registrar cuántas rondas se jugaron

ÚLTIMO ACTO: CICLOS **WHILE** AND **FOR**

Ciclos

- El **while** permite repetir una serie de instrucciones **mientras se cumpla** una condición.
- Si, desde el principio, sabemos que el rango del ciclo es **fijo**, se puede usar **for**.

Ciclos

- El **while** permite repetir una serie de instrucciones **mientras se cumpla** una condición.
- Si, desde el principio, sabemos que el rango del ciclo es **fijo**, se puede usar **for**.
- Definamos la función `suma_elem`, que suma todos los elementos de una lista, usando **while** y **for**:

Con **while**:

```
def suma_elem(listita):  
    suma = 0
```

Ciclos

- El **while** permite repetir una serie de instrucciones **mientras se cumpla** una condición.
- Si, desde el principio, sabemos que el rango del ciclo es **fijo**, se puede usar **for**.
- Definamos la función `suma_elem`, que suma todos los elementos de una lista, usando **while** y **for**:

Con **while**:

```
def suma_elem(listita):  
    suma = 0  
    i = 0  
    while i < len(listita):  
        suma = suma + listita[i]  
        i = i + 1  
    return suma
```

Con **for**:

```
def suma_elem(listita):  
    suma = 0
```

Ciclos

- El **while** permite repetir una serie de instrucciones **mientras se cumpla** una condición.
- Si, desde el principio, sabemos que el rango del ciclo es **fijo**, se puede usar **for**.
- Definamos la función `suma_elem`, que suma todos los elementos de una lista, usando **while** y **for**:

Con **while**:

```
def suma_elem(listita):
    suma = 0
    i = 0
    while i < len(listita):
        suma = suma + listita[i]
        i = i + 1
    return suma
```

Con **for**:

```
def suma_elem(listita):
    suma = 0
    for i in range(0, len(listita)
        , 1):
        suma = suma + listita[i]
    return suma
```


Ciclos

- El **while** permite repetir una serie de instrucciones **mientras se cumpla** una condición.
- Si, desde el principio, sabemos que el rango del ciclo es **fijo**, se puede usar **for**.
- Definamos la función `suma_elem`, que suma todos los elementos de una lista, usando **while** y **for**:

Con **while**:

```
def suma_elem(listita):
    suma = 0
    i = 0
    while i < len(listita):
        suma = suma + listita[i]
        i = i + 1
    return suma
```

Con **for**:

```
def suma_elem(listita):
    suma = 0
    for i in range(0, len(listita), 1):
        suma = suma + listita[i]
    return suma
```

range(inf, sup, paso)

Va dando los números desde `inf` hasta `sup` de a `paso`. Si no se los escribe, `inf` vale 0 y `paso` vale 1.

Ciclos: escribimos ahora `dame_chance` con **for**

Definamos la función `dame_chance(resultados, cantidad_maxima)` usando **for** para recorrer los valores de `resultados`

```
def dame_chance(resultados, cantidad_maxima):  
    cuantos = 0  
    for i in range(0, len(resultados), 1):  
        if resultados[i] <= cantidad_maxima:  
            cuantos = cuantos + 1  
    salida = cuantos / len(resultados)  
    return salida
```

```
range(inf, sup, paso)
```

Va dando los números desde `inf` hasta `sup` de a `paso`. Si no se los escribe, `inf` vale 0 y `paso` vale 1.

¡A PROGRAMAR!