

FACULTAD DE  
INGENIERÍA Y CIENCIAS



**UAI**  
UNIVERSIDAD ADOLFO IBÁÑEZ

# INTRODUCCIÓN A PYTHON

## CONTROL DE FLUJO (CICLOS)

Miguel Carrasco  
miguel.carrasco@uai.cl

- ▶ Estructuras de control
  - Ciclos anidados



```
self.FidValue = OrderedDict(sorted(self.items(), key=lambda item: item[0]))  
#Read item in dictionary  
for key, value in item.FidValue.items():  
    typeOfFID = mapFidType.get(key)  
    if (typeOfFID == "DATE"):  
        d = datetime.datetime.strptime(str(value), "%Y-%m-%d")  
        dataCal = datetime.date.strptime(str(value), "%Y-%m-%d")  
        FidAndValue = FidAndValue + value  
    else: FidAndValue = FidAndValue + value
```

```
try:  
    start = date(int(self.start_year.get(self.start_year.index(self.start_year)),  
                int(self.start_day.get(self.start_day.index(self.start_day)),  
                int(self.start_month.get(self.start_month.index(self.start_month))))  
  
    end = date(int(self.end_year.get(self.end_year.index(self.end_year)),  
              int(self.end_day.get(self.end_day.index(self.end_day)),  
              int(self.end_month.get(self.end_month.index(self.end_month))))
```



Cuando tenemos un ciclo, la operación se realiza según alguna condición de término

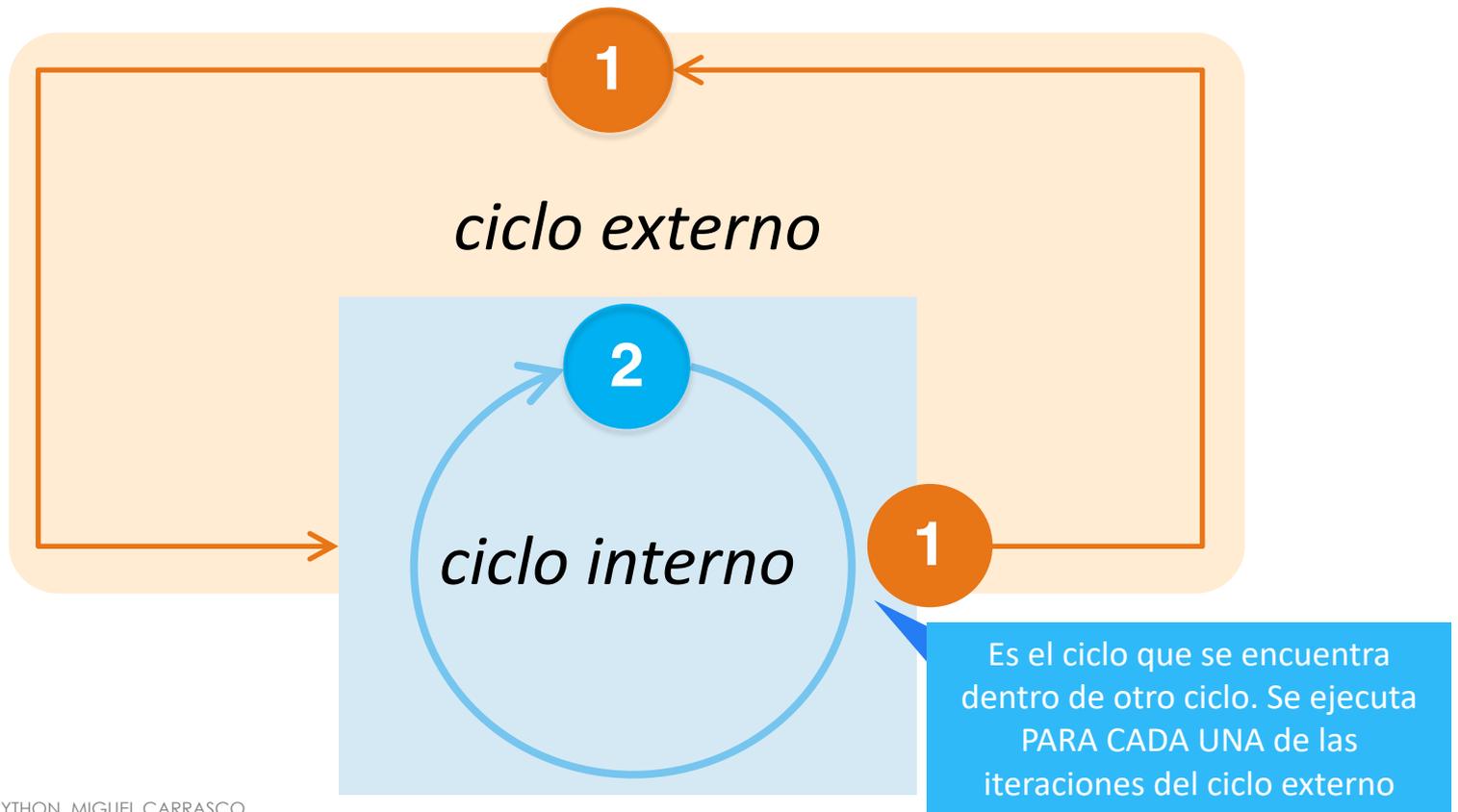
1

*ciclo o loop*

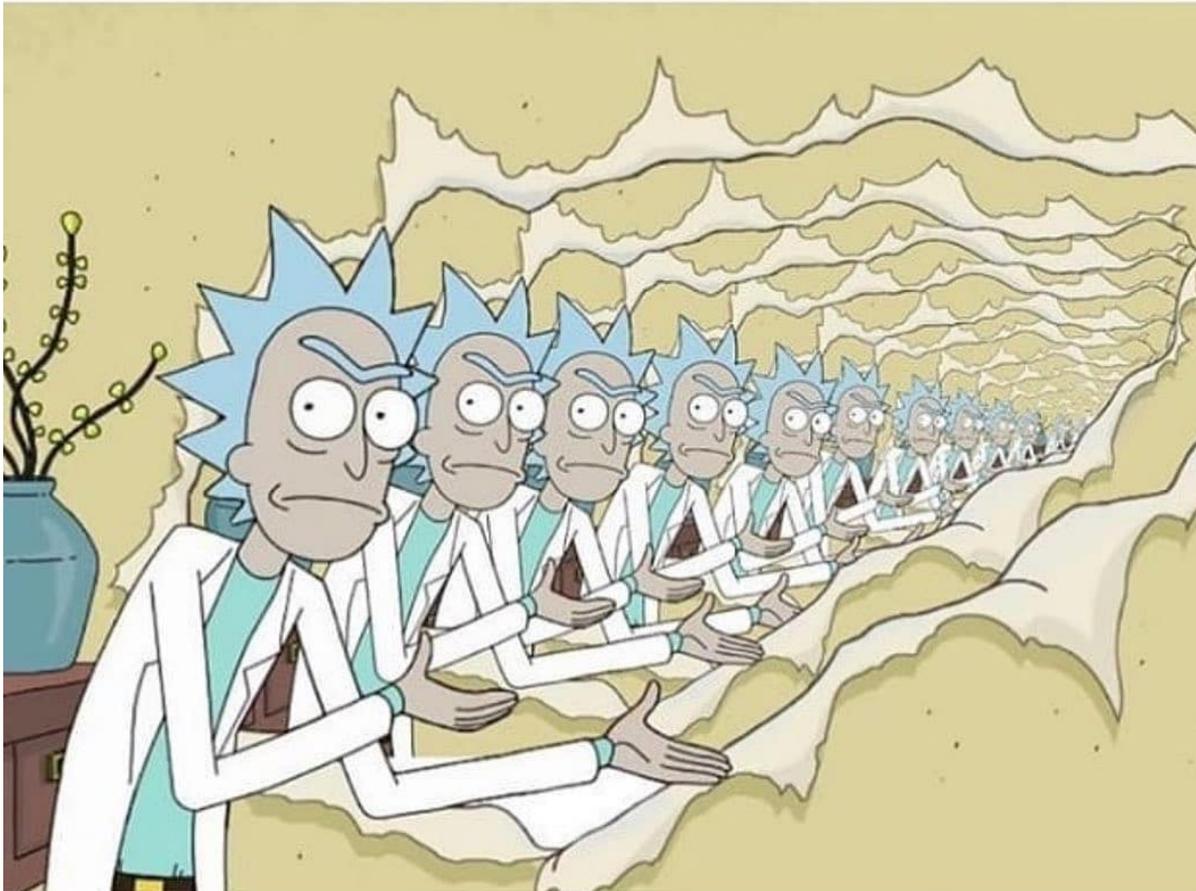
dentro del ciclo loop se encuentran las instrucciones que se repiten mientras dure el ciclo



Cuando hay (al menos) un ciclo dentro de otro ciclo, se habla de ciclos anidados.



## When you forget to break out of the while loop





Estructura:

▶ ciclo externo (se ejecuta  $n$  veces):

▶ ciclo interno (se ejecuta  $m$  veces):  
instrucciones de ciclo interno

*se ejecuta  $n \times m$  veces,  
si son independientes*



¿Qué aparece en pantalla al ejecutar este algoritmo?

```
i = 1
while i<5:
    j = 1
    while j<5:
        print(i, ",", j)
        j = j+1
    i = i+1
```

```
i = 1  
while i<5:  
    j = 1
```

```
while j<5:  
    print(i, ", ", j)  
    j = j+1
```

```
i = i+1
```

Ciclo interno (comienza con j=1)

```
while j<5:  
    print(i, ", ", j)  
    j = j+1
```



Solo una vez que terminan **todas las iteraciones del ciclo interno** se evalúa si se realiza una siguiente iteración del ciclo externo.

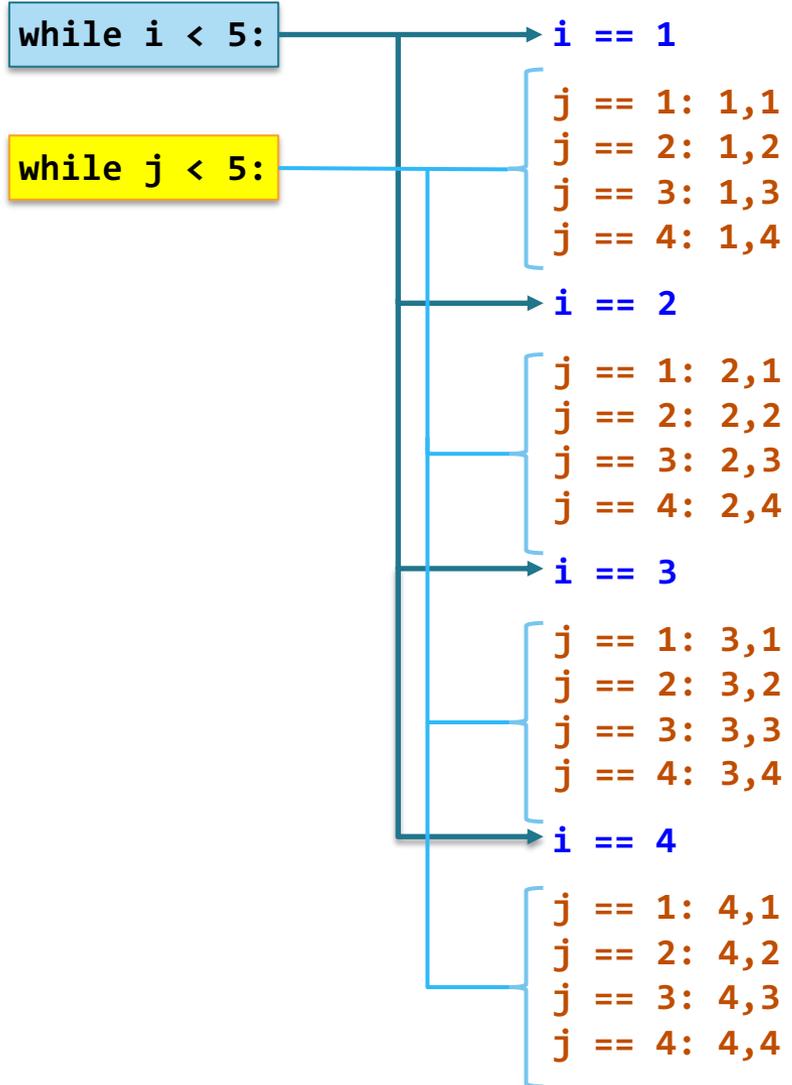
```
i = 1
while i<5:
    j = 1
    while j<5:
        print(i, ", ", j)
        j = j+1
    i = i+1
```

Ciclo externo (se ejecuta 4 veces)

Ciclo interno (se ejecuta 4 veces)

Instrucciones ciclo interno  
(se ejecuta 16 veces)

```
i = 1
while i < 5:
    j = 1
    while j < 5:
        print(i, ", ", j)
        j = j + 1
    i = i + 1
```





Estructura:

▶ ciclo externo (se ejecuta  $n$  veces):

▶ ciclo interno (se ejecuta  $F(n)$  veces):  
instrucciones de ciclo interno

se ejecuta  
 $F(a)+F(b)+\dots+F(n)$  veces,  
ya que  $n$  depende del ciclo  
externo

se ejecuta  $n \times m$  veces,  
si son independientes

¿Qué aparece en pantalla al ejecutar  
este algoritmo?

```
i = 1
while i<5:
    j = 1
    while j<=i:
        print(i, ",", j)
        j = j+1
    i = i+1
```

respuesta

```
1 , 1
2 , 1
2 , 2
3 , 1
3 , 2
3 , 3
4 , 1
4 , 2
4 , 3
4 , 4
```

- ▶ Estructuras de control
  - lista de números (range)
  - ciclo FOR
  - ciclos Anidados
  - Ejercicios



```
self.FidValue = OrderedDict(sorted(self.items(), key=lambda item: item[0]))
#Read item in dictionary
for key, value in item.FidValue.items():
    typeOfFID = mapFidType.get(key)
    if(typeOfFID == "DATE"):
        d = datetime.datetime.strptime(str(value), "%Y-%m-%d")
        dataCal = datetime.date.strptime(str(value), "%Y-%m-%d")
        FidAndValue = FidAndValue + value
    else:FidAndValue = FidAndValue + value
```

```
try:
    start = date(int(self.start_year.get(self.months.index(self.start_month)),
                int(self.start_day.get(self.months.index(self.start_month))),
                int(self.start_year.get(self.months.index(self.start_month))))
    end = date(int(self.end_year.get(self.months.index(self.end_month)),
                int(self.end_day.get(self.months.index(self.end_month))),
                int(self.end_year.get(self.months.index(self.end_month))))
```

- 1) Crea un programa que despliegue el siguiente patrón de números:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

El número de filas lo ingresa el usuario.

- 2) Crea un programa que despliegue el siguiente patrón de números:

```
1
2 3
4 5 6
7 8 9 10
```

El número de filas lo ingresa el usuario.

- 3) Crea un programa que despliegue el valor de la siguiente expresión

$$1^1 + (2^1 + 2^2) + (3^1 + 3^2 + 3^3) + \dots + (N^1 + N^2 + \dots + N^N), \text{ el valor } N \text{ se solicita al usuario.}$$

Por ejemplo, si  $N = 3$  se debería desplegar 46.

- 4) Escribe el código que permite resolver la siguiente serie  $1! + 2! + 3! + \dots + N!$ , el valor de  $N$  se solicita al usuario. Por ejemplo, si  $N = 3$  se debería desplegar 9.

(Ruteo) Escribe lo que aparece en pantalla al ejecutar el siguiente programa en Python:

```
for i in range(0,5):  
    for j in range(0,i):  
        print(i,",",j)
```

Escribe un programa en Python que te permita generar la mitad derecha de un triángulo. El usuario ingresa el alto. Por ejemplo:

```
Ingresa alto piramide:  
5  
#  
##  
###  
####  
#####
```

(Ruteo) ¿Qué aparece en pantalla al ejecutar este programa con datos 3 y 4? ¿y con datos 5 y 3?

```
alto=int(input("Alto del rectangulo:"))  
ancho=int(input("Ancho del rectangulo:"))  
for i in range(1, alto+1):  
    for j in range(1, ancho+1):  
        print("#", end="")  
    print(" ")
```

Escribe un programa en Python que despliegue el avance de los primeros 3 minutos de un cronómetro minuterero. Es decir, que aparezca en pantalla:

```
0:00  
0:01  
0:02  
0:03  
...  
0:59  
1:00  
1:01  
1:02  
...  
1:59  
2:00  
2:01  
...  
2:59  
3:00
```