



Politecnico  
di Torino

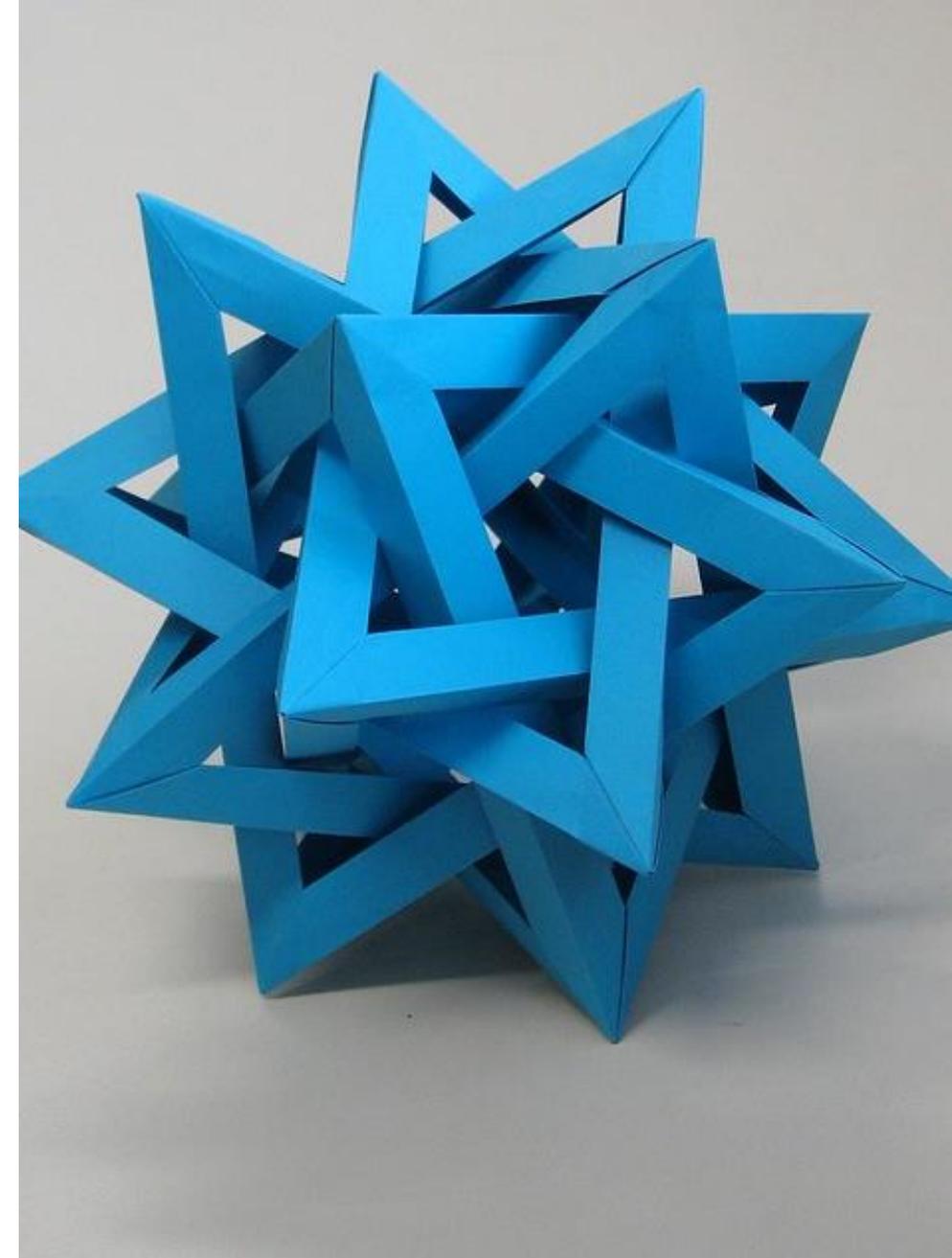
Dipartimento  
di Automatica e Informatica

# Unità P4: Cicli

CICLI, RIPETIZIONI, INTERVALLI, SEQUENZE



Chapter 4



# Obiettivi dell'Unità

- Implementare cicli **while** e **for**
- Fare l'*hand-trace* dell'esecuzione di un programma
- Acquisire familiarità con i tipici algoritmi che utilizzano i cicli
- Comprendere i cicli annidati
- Implementare programmi che leggono e elaborano insiemi di dati
- Utilizzare il computer per simulazioni

# Il ciclo `while`

---

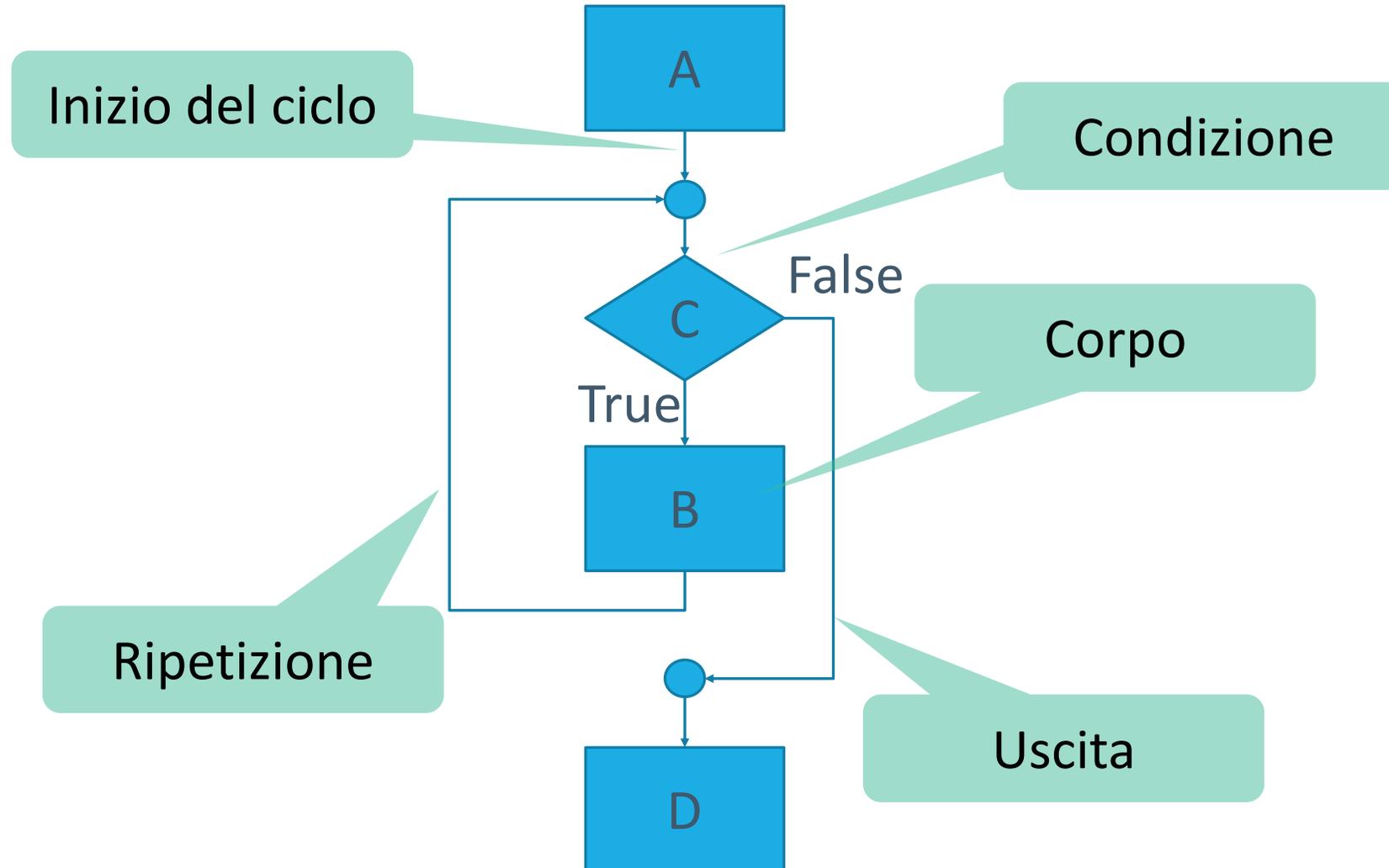


4.1

# Il ciclo `while`

- I cicli sono utilizzati per ripetere diverse volte lo STESSO blocco di istruzioni
  - ... con valori diversi delle variabili
  - ... fino a che una «condizione di ciclo» rimane **Vera** (True)
- Casi d'uso
  - Calcolare un interesse composto
  - Simulazioni, programmi controllati da eventi
  - Analisi di dati simili

# Ciclo *while* - flowchart



# Esempio

- Calcolo di un interesse composto (Capitolo 1)

Iniziare ponendo anno uguale a 0 e saldo uguale a 10000.

Anno	Interessi	Saldo
0		10 000

Ripetere i passi seguenti finché il saldo è minore di 20000.

Aggiungere 1 al valore dell'anno.

Calcolare gli interessi come "saldo per 0.05" (cioè 5 per cento).

Aggiungere gli interessi al saldo.

Riportare il valore finale dell'anno come risposta.



# Pianificare il ciclo `while`

```
balance = 10.0
```

```
TARGET = 100.0
```

```
year = 0
```

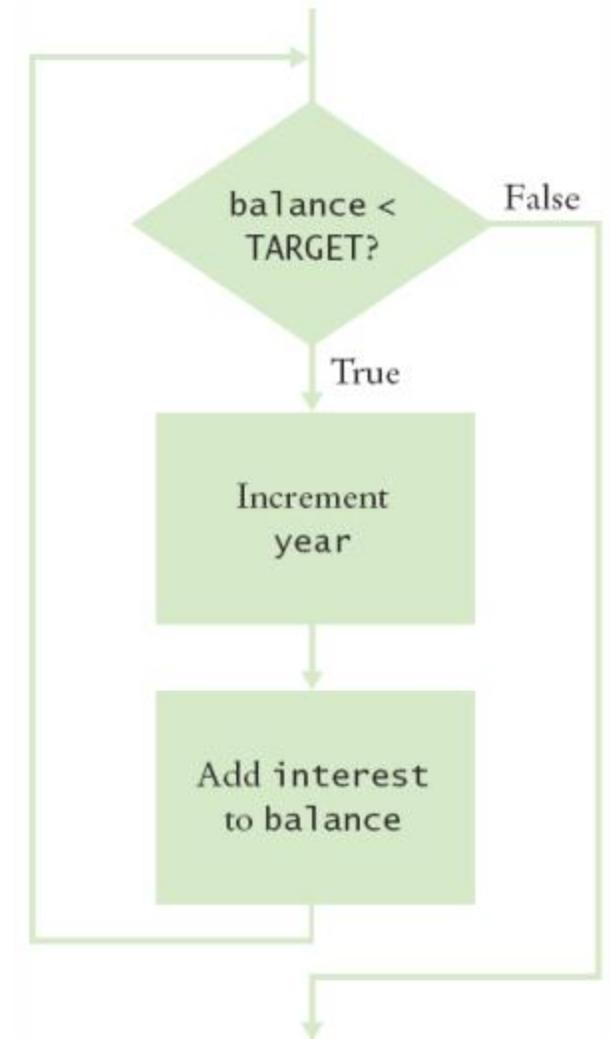
```
RATE = 5
```

```
while balance < TARGET :
```

```
    year = year + 1
```

```
    interest = balance * RATE/100
```

```
    balance = balance + interest
```



- In Python, un ciclo esegue istruzioni FINTANTO CHE LA CONDIZIONE È VERA

# Sintassi: istruzione `while`

## *Sintassi*

```
while condizione :  
    enunciati
```

## *Esempio*

Questa variabile viene inizializzata all'esterno del ciclo e aggiornata al suo interno.

Se la *condizione* non diventa mai falsa, il ciclo è infinito (Errori comuni 4.2).

```
balance = 10000.0  
.  
.  
while balance < TARGET :  
    interest = balance * RATE / 100  
    balance = balance + interest
```

Attenzione agli errori "per scarto di uno" nelle condizioni dei cicli (Errori comuni 4.3).

Ricordarsi i "due punti"!

Questi enunciati vengono eseguiti finché la *condizione* è vera.

Gli enunciati del corpo di un enunciato composto devono iniziare tutti nella stessa colonna, un po' a destra.

# Doubleinv.py

## ch04/doubleinv.py

```
1  ##
2  # This program computes the time required to double an investment.
3  #
4
5  # Create constant variables.
6  RATE = 5.0
7  INITIAL_BALANCE = 10000.0
8  TARGET = 2 * INITIAL_BALANCE
9
10 # Initialize variables used with the loop.
11 balance = INITIAL_BALANCE
12 year = 0
13
14 # Count the years required for the investment to double.
15 while balance < TARGET :
16     year = year + 1
17     interest = balance * RATE / 100
18     balance = balance + interest
19
20 # Print the results.
21 print("The investment doubled after", year, "years.")
```

Dichiara e inizializza una variabile fuori dal ciclo per contare gli anni: `year`

Incrementa la variabile `year` ad ogni iterazione del ciclo

# Esecuzione del ciclo

---

① Verifica della condizione del ciclo

balance =   
year =

```
while balance < TARGET :  
    year = year + 1  
    interest = balance * RATE / 100  
    balance = balance + interest
```

La condizione  
è vera

② Esecuzione degli enunciati interni al ciclo

balance =   
year =   
interest =

```
while balance < TARGET :  
    year = year + 1  
    interest = balance * RATE / 100  
    balance = balance + interest
```

③ Nuova verifica della condizione del ciclo

balance =   
year =   
interest =

```
while balance < TARGET :  
    year = year + 1  
    interest = balance * RATE / 100  
    balance = balance + interest
```

La condizione  
è ancora vera

# Esecuzione del ciclo (2)

④ Dopo 15 iterazioni

balance =   
year =   
interest =

```
while balance < TARGET :  
    year = year + 1  
    interest = balance * RATE / 100  
    balance = balance + interest
```

La condizione  
non è più vera

⑤ Esecuzione dell'enunciato che segue il ciclo

balance =   
year =   
interest =

```
while balance < TARGET :  
    year = year + 1  
    interest = balance * RATE / 100;  
    balance = balance + interest
```

```
print(year)
```

# Cicli controllati da un **contatore**

- Un ciclo `while` controllato da un **contatore**
  - Conta il numero di ripetizioni effettuate
  - Si ferma quando sono state completate un numero predefinito di iterazioni

```
counter = 1                # Inizializza il contatore
while counter <= 10 :     # Controlla il contatore
    print(counter)
    counter = counter + 1  # Aggiorna il contatore
```

# Cicli controllati da un **evento**

- Un ciclo `while` controllato da una condizione che, prima o poi, diventa falsa
  - Il numero di iterazioni non è noto in anticipo

```
balance = INITIAL_BALANCE # Inizializza la variabile del loop
while balance <= TARGET:  # Check the loop variable
    year = year + 1
    balance = balance * 2  # Update the loop variable
```

# Esempio - Investimento

- Aprire il file:
  - Doubleinv.py
- Eseguire il programma ed eseguirlo con diversi casi di test
  - Il programma chiederà di inserire un tasso di interesse
  - Inserire un mix di valori validi e non validi

# Ciclo `while` – Esempi

Ciclo	Visualizza	Spiegazione
<pre>i = 0 total = 0 while total &lt; 10 :     i = i + 1     total = total + i     print(i, total)</pre>	<pre>1 1 2 3 3 6 4 10</pre>	Quando <code>total</code> diventa uguale a 10, la condizione del ciclo diventa falsa e il ciclo termina.
<pre>i = 0 total = 0 while total &lt; 10 :     i = i + 1     total = total - i     print(i, total)</pre>	<pre>1 -1 2 -3 3 -6 4 -10 . . .</pre>	Dato che <code>total</code> non diventa mai uguale a 10, si ha un ciclo infinito (Errori comuni 4.2)
<pre>i = 0 total = 0 while total &lt; 0 :     i = i + 1     total = total - i     print(i, total)</pre>	(Niente)	La condizione <code>total &lt; 0</code> è già falsa alla prima verifica, per cui il ciclo non viene mai eseguito.

# Ciclo `while` – Esempi (2)

Ciclo	Visualizza	Spiegazione
<pre>i = 0 total = 0 while total &gt;= 10 :     i = i + 1     total = total + i     print(i, total)</pre>	(Niente)	Probabilmente il programmatore voleva scrivere “fermati quando la somma vale almeno 10”, però la condizione presente nel ciclo dice quando il corpo va eseguito, non quando il ciclo deve terminare (Errori comuni 4.1).
<pre>i = 0 total = 0 while total &gt;= 0 :     i = i + 1     total = total + i     print(i, total)</pre>	(Niente, il programma non termina)	Dato che <code>total</code> è e sarà sempre maggiore o uguale a zero, il ciclo prosegue all’infinito. Non visualizza niente perché la funzione <code>print</code> viene invocata fuori dal corpo del ciclo, dato che non ne fa parte (si osservi l’incollamento a sinistra).

# Errore comune: condizione di test

- Il corpo del ciclo viene eseguito se la condizione è **True**.
- Si assuma che **bal** sia inizializzata a un valore inferiore a TARGET e che debba crescere fino a diventare uguale a TARGET
  - Quale versione del ciclo è corretta?

```
while bal >= TARGET :  
    year = year + 1  
    interest = bal * RATE  
    bal = bal + interest
```

```
while bal < TARGET :  
    year = year + 1  
    interest = bal * RATE  
    bal = bal + interest
```

# Errore comune: Cicli infiniti

- Il ciclo viene eseguito fino a che la condizione non diventa **falsa**.
- Cosa succede se NEL CICLO mi dimentico di aggiornare la variabile su cui viene fatto il test?
  - `bal` è la variabile testata (TARGET non cambia)
  - Si resta nel ciclo all'infinito (o fino a che non si interrompe forzatamente il programma)

```
while bal < TARGET :  
    year = year + 1  
    interest = bal * RATE  
    bal = bal + interest
```

# Errore comune: Off-by-One Errors

- Una variabile 'contatore' viene usata nella condizione
- Il contatore può essere inizializzato a 0 o a 1, ma in informatica un contatore si inizializza solitamente a 0
- Se voglio dipingere tutte e 5 le dita della mano, quando mi devo fermare?
  - Se inizio da 0, si usa '<'
  - 0, 1, 2, 3, 4

Se inizio da 1, si usa '<='

1, 2, 3, 4, 5

```
finger = 0
FINGERS = 5
while finger < FINGERS :
    # paint finger
    finger = finger + 1
```

```
finger = 1
FINGERS = 5
while finger <= FINGERS :
    # paint finger
    finger = finger + 1
```

# Cicli misti

- Non tutti i cicli sono controllati solamente da un contatore o da un evento
- Esempio: leggi una sequenza di numeri. Il programma interrompa la lettura quando ho letto 10 numeri oppure quando leggo il numero 0

```
count = 0
ok = True

while (count < 10) and ok :
    a = int(input('Number: '))
    if a==0:
        ok = False
    print(f'Number {count+1}={a}')
    count = count + 1
```

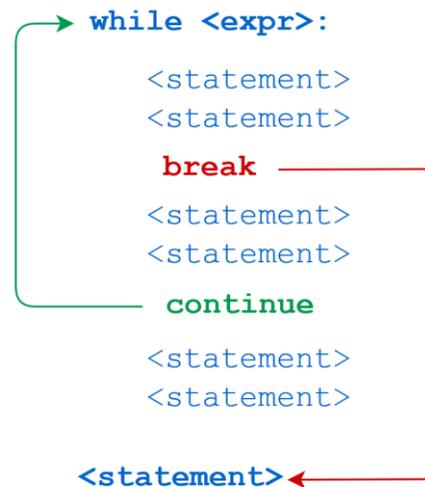
# Interruzione anticipata di un ciclo

## BREAK

- Se all'interno di un ciclo si inserisce l'istruzione **break**
  - il ciclo viene interrotto
  - senza completare l'iterazione corrente
  - anche se non sono state fatte tutte le iterazioni necessarie
  - l'esecuzione continua con le istruzioni successive al ciclo

## CONTINUE

- Se all'interno di un ciclo si inserisce l'istruzione **continue**
  - Non viene completata l'iterazione corrente
  - Il ciclo continua normalmente con l'iterazione successiva (senza terminare quella corrente)



<https://realpython.com/python-while-loop/>

# *Hand-Tracing* di cicli

---



4.2

# Hand-Tracing di cicli

- Esempio: Calcolare la somma di cifre numeriche (1+7+2+9)
  - Creare una colonna per ogni variabile importante (n, total, digit)
  - Esaminare il codice ed identificare i vari blocchi
  - Inizializzare le variabili prima del ciclo

```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10
```



n	total	digit
1729	0	

```
print(total)
```

# Tenere traccia della somma di cifre

n	total	digit
1729	0	

```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```

- Si inizi ad eseguire il corpo del ciclo aggiornando il valore delle variabili in una nuova riga
- Si cancellino i valori nelle righe precedenti

# Tenere traccia della somma di cifre

n	total	digit
1729	<del>0</del>	
	9	9

```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```

- Continuare ad eseguire i cicli aggiornando le variabili
- $1729 // 10$  fa 172 (senza resto)

# Tenere traccia della somma di cifre

- Test della condizione: se Vera, si esegue di nuovo il ciclo
  - La variable n è 172; 172 è > 0?, Vero!
- Creo una nuova riga nella tabella per aggiornare le variabili

n	total	digit
<del>1729</del>	<del>0</del>	
<del>172</del>	<del>9</del>	<del>9</del>
17	11	2

```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```

# Tenere traccia della somma di cifre

- Terza iterazione
  - La variabile n è 17 che è ancora maggiore di 0
- Eseguo il ciclo e aggiorno di nuovo le variabili

n	total	digit
<del>1729</del>	<del>0</del>	
<del>172</del>	<del>9</del>	<del>9</del>
<del>17</del>	<del>11</del>	<del>2</del>
1	18	7

```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```

# Tenere traccia della somma di cifre

- Quarta iterazione:

- La variabile n è 1 all'inizio del ciclo.  $1 > 0$ ? True
- Eseguo il ciclo e aggiorno la variabile n a 0 ( $1/10 = 0$ )

n	total	digit
<del>1729</del>	<del>0</del>	
<del>172</del>	<del>9</del>	<del>9</del>
<del>17</del>	<del>11</del>	<del>2</del>
<del>1</del>	<del>18</del>	<del>7</del>
0	19	1



```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```

# Tenere traccia della somma di cifre

- Dato che  $n$  è 0, l'espressione  $(n > 0)$  è **Falsa**
- Il ciclo non è più eseguito
  - Si salta alla prima istruzione dopo il corpo del ciclo
- Viene finalmente stampata la somma!

n	total	digit	output
<del>1729</del>	<del>0</del>		
<del>172</del>	<del>9</del>	<del>9</del>	
<del>17</del>	<del>11</del>	<del>2</del>	
<del>7</del>	<del>18</del>	<del>7</del>	
0	19	1	19

```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```



# Sommario del ciclo `while`

- I cicli `while` sono molto comuni
- Si devono inizializzare le variabili PRIMA di controllarle
  - La condizione è **testata PRIMA** del corpo del ciclo
    - La condizione spesso utilizza una variabile **contatore**
- Attenzione ai cicli infiniti!
  - All'interno del ciclo qualche variabile coinvolta nella condizione di uscita **DEVE CAMBIARE**
  - Questo cambio deve far sì che, prima o poi, la condizione diventi **Falsa**

# Valori sentinella

---



4.3

# Lavorare con Valori «Sentinella»

- Quando si legge una sequenza di input, è spesso necessario un metodo per **indicare la fine della sequenza**
- Quando non si sa quanti elementi ci sono in una lista, si può utilizzare un carattere o un valore speciale per **identificare l'ultimo della lista**
  - Questo valore si chiama “**sentinella**”
  - Per input di numeri interi positivi, si può ad esempio usare 0 o -1
  - Una sentinella identifica la fine dei dati ma **NON FA PARTE DEI DATI STESSI**

```
salary = 0.0
while salary >= 0 :
    salary = float(input())
    if salary >= 0.0 :
        total = total + salary
        count = count + 1
```

# Algoritmo: Media di un insieme di valori

- Dichiarare e inizializzare
  - una variabile 'total' a 0 , una variabile 'count' a 0 , una variabile 'salary' a 0
- Stampare le istruzioni per l'input dati (es. «inserisci dei numeri positivi e termina con -1»)
- Impostare un ciclo di input che termina alla lettura del valore sentinella (cioè un valore inferiore a 0)
  - Salva il valore in input nella variabile '**salary**'
  - Se **salary** è un valore positivo o nullo
    - Aggiungi la variabile **salary** a **total**
    - Incrementa di 1 la variabile **count**
- Assicurarsi che ci sia almeno un valore ( $\text{count} \neq 0$ ) prima di fare la divisione
  - Dividere **total** per **count** e stampare il risultato.

# Sentinel.py (1)

```
5 # Initialize variables to maintain the running total and count.
```

```
6 total = 0.0
```

```
7 count = 0
```

Fuori dal ciclo while: dichiarazione ed inizializzazione delle variabili da utilizzare

```
8
```

```
9 # Initialize salary to any non-sentinel value.
```

```
10 salary = 0.0
```

```
13 while salary >= 0.0 :
```

Dato che salary è inizializzata a 0, il ciclo while verrà eseguito almeno una volta

```
14     salary = float(input("Enter a salary or -1 to finish: "))
```

```
15     if salary >= 0.0 :
```

Inserire nuovo salary e confrontarlo con il valore sentinella

```
16         total = total + salary
```

```
17         count = count + 1
```

Aggiornare total e count (per calcolare la media più avanti)

# Sentinel.py (2)

```
19 # Compute and print the average salary.
20 if count > 0 :      Evita una divisione per 0
21     average = total / count
22     print("Average salary is", average)

23 else :
24     print("No data was entered.")
```

Calcolare e stampare la media di salary usando le variabili total e count

## Program Run

```
Enter salaries, -1 to finish: 10 10 40 -1
Average salary: 20
```

# Esempio Sentinella

- Aprire il file:
  - Sentinel.py
- Notare l'utilizzo dell' **if** all'interno del ciclo while
  - L' **if** viene usato per assicurarsi di non processare il valore sentinella

# Prima Lettura

- Alcuni programmatori non amano il “trucco” di inizializzare la variabile di input ad un valore diverso dalla sentinella
  - Aumenta la confusione tra valori “legittimi” e “invalidi” della sentinella

```
# Set salary to a value to ensure that the loop
# executes at least once.
salary = 0.0
while salary >= 0 :
```

- Un’alternativa (da usare!!!!) è di precaricare la variabile con una lettura iniziale FUORI DAL CICLO

```
salary = float(input("Enter a salary or -1 to finish: "))
while salary >= 0 :
```

# Letture seguente

- A questo punto, all'interno del ciclo, la lettura del prossimo valore va spostata alla fine del ciclo.

```
# Priming read
salary = float(input("Enter a salary or -1 to finish: "))
while salary >= 0.0 :
    total = total + salary
    count = count + 1
    # Modification read
    salary = float(input("Enter a salary or -1 to finish: "))
```

# Variabili booleane e Sentinelle

- Una variabile booleana può essere utilizzata per controllare un ciclo
  - In questo caso si chiama variabile *'flag'*

```
done = False
while not done :
    value = float(input("Enter a salary or -1 to finish: "))
    if value < 0.0:
        done = True
    else :
        # Process value
```

Inizializzare **done** in modo che si entri nel ciclo

Settare il 'flag' **done** a Vero se si trova la sentinella

# Semplici algoritmi con cicli



4.5

# Semplici algoritmi con cicli

- Somma e Media
- Conteggio di elementi
- Input fino a che si trova una corrispondenza
- Massimo e Minimo
- Confronto di valori consecutivi

# Somma di valori

- Inizializzare **total** a 0
- Usare il ciclo `while` con una sentinella

```
total = 0.0
inputStr = input("Enter value: ")
while inputStr != "" :
    value = float(inputStr)
    total = total + value
    inputStr = input("Enter value: ")
```

# Media di 'count' valori

- Innanzitutto calcolare la somma dei valori
- Inizializzare **count** a 0
  - Incrementare **count** a ogni input
- Controllare che **count** > 0
  - Dividere la somma per count!

```
total = 0.0
count = 0
inputStr = input("Enter value: ")
while inputStr != "" :
    value = float(inputStr)
    total = total + value
    count = count + 1
    inputStr = input("Enter value: ")

if count > 0 :
    average = total / count
else :
    average = 0.0
```

# Contare elementi (e.g., Numeri negativi)

- Contare elementi
  - Inizializzare **negatives** a 0
  - Utilizzare un ciclo while
  - Incrementare **negatives** se trovo un numero negativo



```
negatives = 0
inputStr = input("Enter value: ")
while inputStr != "" :
    value = int(inputStr)
    if value < 0 :
        negatives = negatives + 1
    inputStr = input("Enter value: ")

print("There were", negatives,
      "negative values.")
```

# Input fino a che si trova un match

- Inizializzare il flag booleano a False
- Testare il valore sentinella nel ciclo while
  - Leggi l'input, e confrontalo con l'intervallo di interesse
    - Se **input** è nell'intervallo di interesse, cambia il flag a Vero
    - Il ciclo si interromperà per via del nuovo valore del flag

```
valid = False
while not valid :
    value = int(input("Please enter a positive value < 100: "))
    if value > 0 and value < 100 :
        valid = True
    else :
        print("Invalid input.")
```

***Questo è un ottimo modo di validare gli input***

# Massimo

- Leggere il primo valore
  - Per definizione, questo è il valore più grande visto fino a quel momento
- Il ciclo continua finché si leggono numeri validi (non-sentinella)
  - Leggere un nuovo valore
  - Confrontarlo con il più grande (**largest**)
  - Se necessario aggiornare **largest**

```
largest = int(input("Enter a value: "))
inputStr = input("Enter a value: ")
while inputStr != "" :
    value = int(inputStr)
    if value > largest :
        largest = value
    inputStr = input("Enter a value: ")
```

# Minimo

- Leggere il primo valore
  - Per definizione, questo è il valore più piccolo visto fino a quel momento
- Il ciclo continua finché si leggono numeri validi (non-sentinella)
  - Leggere un nuovo valore
  - Confrontarlo con il più piccolo (**smallest**)
  - Se necessario aggiornare **smallest**

```
smallest = int(input("Enter a value: "))
inputStr = input("Enter a value: ")
while inputStr != "" :
    value = int(inputStr)
    if value < smallest :
        smallest = value
    inputStr = input("Enter a value: ")
```

# Confronto di valori consecutivi

- Leggere il primo valore
- Utilizzare un `while` per valutare gli input successivi e scartare valori consecutivi uguali
  - Copiare `value` nella variabile `previous`
  - Leggere il `valore successivo` nella variabile `value`
  - Confrontare `value` e `previous`, e stampare un messaggio se sono uguali

```
value = int(input("Enter a value: ")) # 1st
inputStr = input("Enter a value: ")
while inputStr != "" :
    previous = value
    value = int(inputStr)
    if value == previous :
        print("Duplicate input")
    inputStr = input("Enter a value: ")
```

# Esempio - Voti

- Aprire il file:
  - Grades.py
- Analizzare con attenzione il codice
- Il massimo punteggio possibile è letto come input
  - C'è un ciclo per validare questo input
- Il voto finale è calcolato come il 60% dei punti disponibili

# Il ciclo for

---



4.6

# Il ciclo `for...in`

- Il ciclo `for...in` serve a **iterare su tutti i valori** di un qualsiasi **contenitore**
- Un contenitore è un oggetto (come una stringa) che può memorizzare un insieme di elementi
- Python ha diversi tipi di contenitori
  - Una stringa è un contenitore di caratteri
  - Una **lista** (Capitolo 6) è un contenitore di valori arbitrari (numerici stringhe, liste, ...)
  - Un **file** (Capitolo 7) è un contenitore di linee di testo

# Esempio di un ciclo **for**

```
stateName = "Virginia"
i = 0
while i < len(stateName) :
    letter = stateName[i]
    print(letter)
    i = i + 1
```

Versione while

```
stateName = "Virginia"
for letter in stateName :
    print(letter)
```

Versione for

- Notare una differenza importante tra i cicli **while** e **for**.
- Nel ciclo **while**, alla variabile indice (i) vengono assegnati i valori 0, 1, .... E così via.
- Nel ciclo **for**, alla variabile **letter** vengono assegnati tutti gli elementi del contenitore:
  - stateName[0], stateName[1], e così via.
- Nota: "Virginia" è una stringa, contenitore della sequenza di lettere "V", "i", "r", "g", "i", "n", "i", "a"

# Il contenitore 'range'

- È un contenitore speciale di una **sequenza di numeri** consecutivi.
- Si genera con la funzione **range(N)**
  - **range(N)** crea una sequenza di interi da **0** a **N-1**
- Quindi un **for** con un **range()** è equivalente a un ciclo **while** controllato da un contatore

# Il ciclo **for** (2)

- Si può utilizzare un ciclo **for** per creare un ciclo controllato da un contatore.

```
i = 1
while i < 10 :
    print(i)
    i = i + 1
```

Versione while

```
for i in range(1, 10) :
    print(i)
```

Versione for

# Sintassi di un'istruzione for (*Contenitore*)

- Si usa un ciclo per iterare sul contenuto di un contenitore, un elemento alla volta

## ***Sintassi***

```
for variabile in contenitore :  
    enunciati
```

## ***Esempio***

A questa variabile viene assegnato un valore ad ogni iterazione del ciclo.

```
for letter in stateName :  
    print(letter)
```

Un contenitore.

La variabile contiene un elemento, non un indice.

Gli enunciati del corpo del ciclo vengono eseguiti per ciascun elemento del contenitore.

# Sintassi di un'istruzione for (*Range*)

- Si può utilizzare un **for** controllato da un range per iterare su di un insieme di valori interi
- La funzione **range** genera una sequenza di interi che possono essere usati all'interno del ciclo

**Sintassi**

```
for variabile in range(...):  
    enunciati
```

---

**Esempi**

All'inizio di ciascuna iterazione, a questa variabile viene assegnato il valore successivo della sequenza generata dalla funzione range.

La funzione range genera una sequenza di numeri interi che controlla le iterazioni del ciclo.

Con un solo argomento, la sequenza inizia da 0, con incrementi unitari, e l'argomento è il primo valore che NON appartiene alla sequenza.

```
for i in range(5):  
    print(i) # Visualizza 0, 1, 2, 3, 4
```

Con due argomenti, la sequenza inizia dal primo argomento.

```
for i in range(1, 5):  
    print(i) # Visualizza 1, 2, 3, 4
```

Con tre argomenti, il terzo argomento è il valore dell'incremento.

```
for i in range(1, 11, 2):  
    print(i) # Visualizza 1, 3, 5, 7, 9
```

# Esempi di for corretti

- Create cicli semplici!

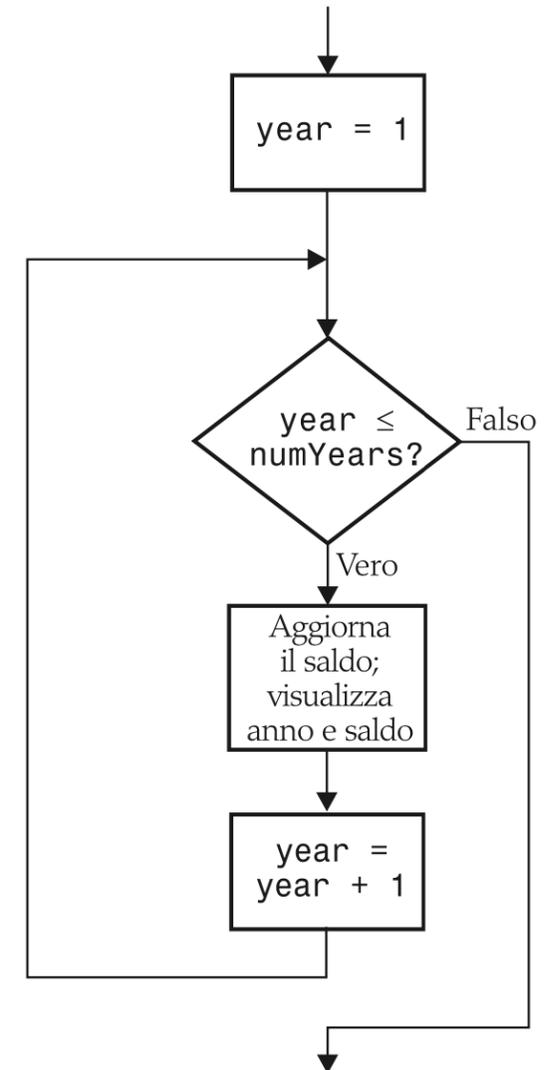
<b>Ciclo</b>	<b>Valori di i</b>	<b>Commento</b>
<code>for i in range(6) :</code>	0, 1, 2, 3, 4, 5	Il ciclo viene eseguito 6 volte.
<code>for i in range(10, 16) :</code>	10, 11, 12, 13, 14, 15	Il valore finale non appartiene alla sequenza.
<code>for i in range(0, 9, 2) :</code>	0, 2, 4, 6, 8	Il terzo argomento è l'incremento tra un valore e il successivo.
<code>for i in range(5, 0, -1) :</code>	5, 4, 3, 2, 1	Per contare a ritroso si usa un incremento negativo.

# Attenzione ai parametri di range()

- Stampare il netto alla fine di ogni anno per un certo numero di anni

Year	Balance
1	10500.00
2	11025.00
3	11576.25
4	12155.06
5	12762.82

```
for year in range(1, numYears + 1) :  
    Update balance.  
    Print year and balance.
```



# Esempio: investimenti

```
1  ##
2  # This program prints a table showing the growth of an investment.
3  #
4
5  # Define constant variables.
6  RATE = 5.0
7  INITIAL_BALANCE = 10000.0
8
9  # Obtain the number of years for the computation.
10 numYears = int(input("Enter number of years: "))
11
12 # Print the table of balances for each year.
13 balance = INITIAL_BALANCE
14 for year in range(1, numYears + 1) :
15     interest = balance * RATE / 100
16     balance = balance + interest
17     print("%4d %10.2f" % (year, balance))
```

# Suggerimento

- Trovare il limite inferiore e superiore di iterazioni di un ciclo può sembrare complesso.
  - Devo partire da  $0$  o da  $1$ ?
  - Devo usare  $\leq b$  o  $< b$  come condizione di terminazione?
- Il conteggio è più semplice se il ciclo ha dei limiti asimmetrici
  - I seguenti cicli sono eseguiti  $b-a$  volte
  - Il valore di  $a$  è **incluso**, quello di  $b$  **escluso**.

```
int i = a
while i < b :
    . . .
    i = i + 1
```

```
for i in range(a, b) :
    . . .
```

# Suggerimento

- Un ciclo con limiti simmetrici (" $\leq$ "), è eseguito  **$b-a+1$**  volte.
  - Quel "+1" è una sorgente di molti errori!

```
i = a
while i <= b :
    . . .
    i = i + 1
```

```
# For this version of the loop the
'+1' is very noticeable!
for year in range(1, numYears + 1) :
```

# Sommario del ciclo **for**

- I cicli **for** sono molto versatili
- Il ciclo **for** si utilizza per iterare il contenuto di un qualunque contenitore, che è un oggetto che memorizza un insieme di elementi
  - Una stringa è un contenitore che memorizza un insieme di caratteri.
- Un ciclo **for** può anche essere usato per realizzare un ciclo controllato da un contatore, e farlo quindi iterare su un insieme di valori interi tramite la funzione **range**.

# Passi per scrivere un ciclo

## ■ Progetto:

- Decidere cosa si deve fare all'interno del ciclo
- Individuare la condizione da valutare
- Determinare il tipo di ciclo più opportuno (for o while)
- Inizializzare le variabili prima del ciclo
- Elaborare i risultati dopo che il ciclo è terminato
- Simulare (hand-tracing) il ciclo usando dati di esempio

## ■ Codifica:

- Implementare il ciclo in Python

# Ciclare su indice e valore

---

# Stampiamo i caratteri ed il loro indice

WHILE -> CICLO SULL'INDICE

```
i = 0
while i < len(nome):
    lettera = nome[i]
    print(i, lettera)
    i = i + 1
```

Scomodo gestire  
l'indice  
manualmente

FOR -> CICLO SUL VALORE

```
for lettera in nome:
    print(lettera)
    # non ho il valore di 'i'
```

Breve e compatto,  
ma non ho il valore  
dell'indice

```
i = 0
for lettera in nome:
    print(lettera)
    i = i + 1
```

Devo ricalcolarmi  
l'indice, scomodo  
come il while

# Stampiamo i caratteri ed il loro indice

WHILE -> CICLO SULL'INDICE

FOR -> CICLO SUL VALORE

ENUMERATE -> **INDICE E VALORE INSIEME**

```
for (i, lettera) in enumerate(nome):  
    print(i, lettera)
```

La funzione `enumerate` restituisce, ad ogni iterazione, **una coppia** di valori (indice, valore)

Il ciclo **for** può «estrarre» entrambi i valori, che si potranno utilizzare nel corpo dell'iterazione

:  
ore di 'i'

Breve e compatto,  
ma non ho il valore  
dell'indice

:  
devo ricalcolarmi  
l'indice, scomodo  
come il while

# La funzione `enumerate()`

- La funzione `enumerate` trasforma un contenitore in qualsiasi contenitore in una *sequenza di coppie*<sup>(\*)</sup>

- `nome = "ciao"`



- `enumerate(nome)`



- `for (indice, valore) in enumerate(nome):`

(\*) nell'unità P6 impareremo che si tratta di una *lista di tuple*

# Cicli annidati

---

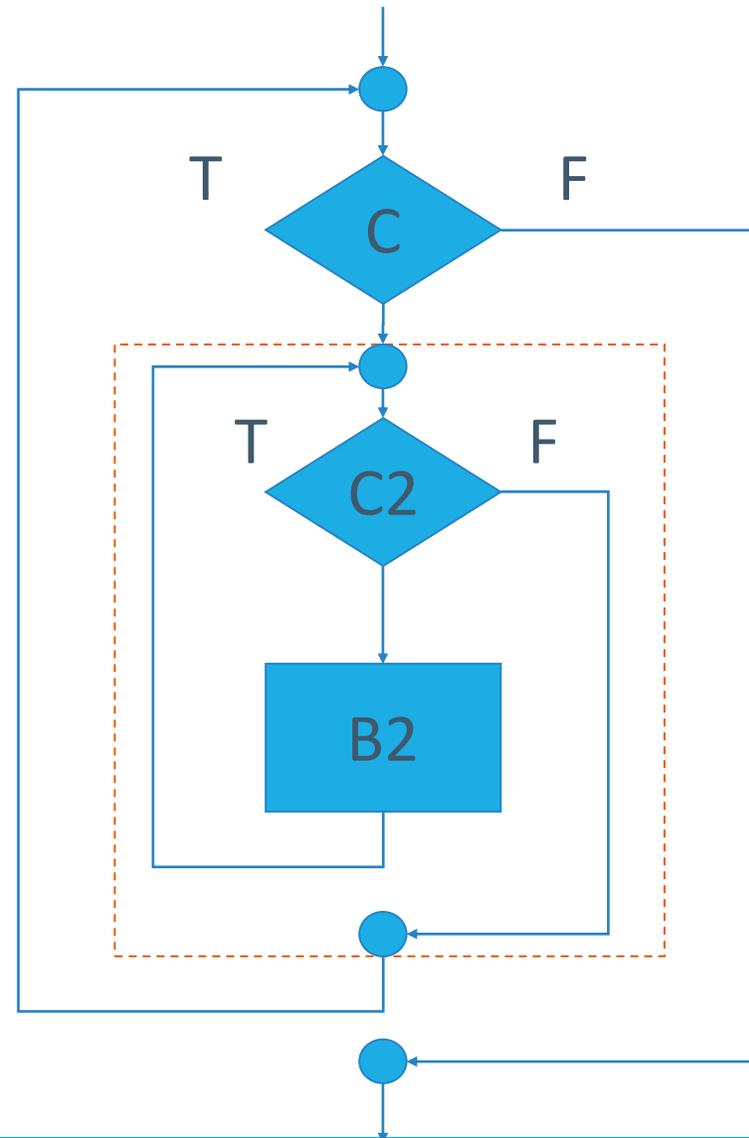
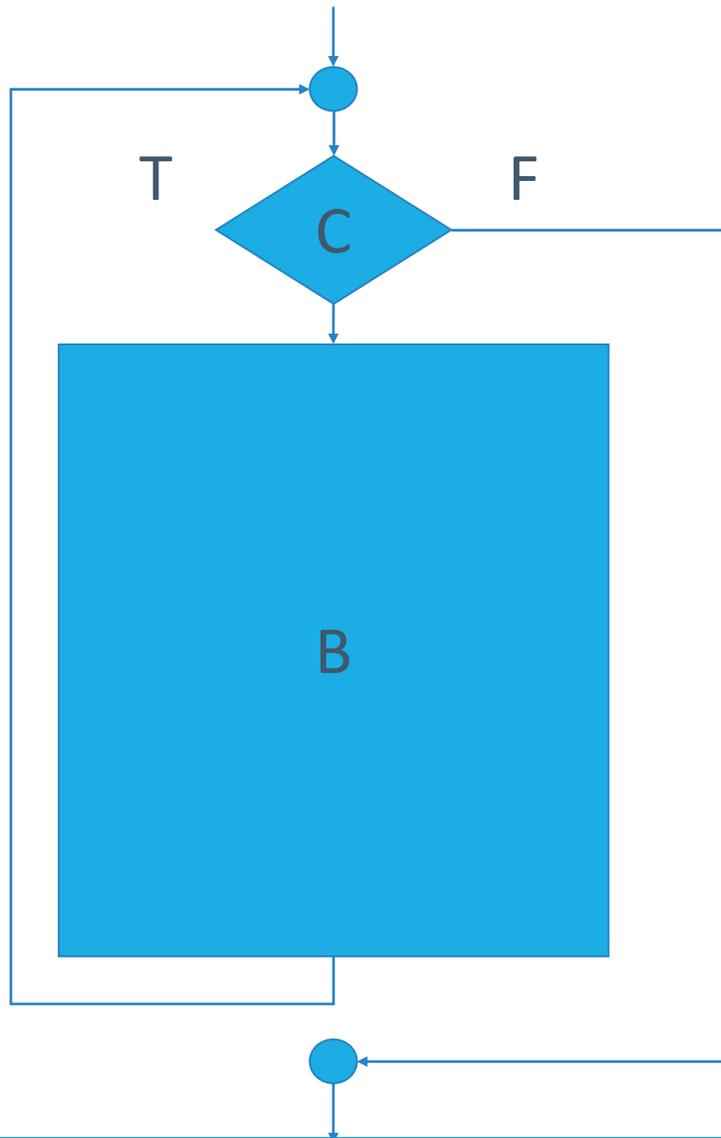


4.7

# Cicli all'interno di cicli

- Abbiamo già imparato ad annidare istruzioni if per prendere delle decisioni complesse.
  - Ricorda: ogni volta che si annida un blocco, questo va **indentato** di un livello
- Problemi più complessi richiedono spesso di annidare un ciclo **annidare un ciclo all'interno di un altro**
  - Il ciclo annidato sarà indentato all'interno del blocco di codice dell'altro ciclo.
- Un buon esempio di utilizzo di cicli annidati è l'analisi delle celle di una tabella (o matrice)
  - Il ciclo più esterno itera sulle **righe** della tabella
  - Il ciclo più **interno** processa le colonne **colonne** nella riga corrente

# Cicli **while** annidati



# Esempio - Definizione del problema

- Stampare l'intestazione di una tabella con i valori  $x^1$ ,  $x^2$ ,  $x^3$ , e  $x^4$
- Stampare una tabella con 4 colonne e 10 righe (una per ogni valore di  $x$ ) che contiene le potenze di  $x^1$ ,  $x^2$ ,  $x^3$ , and  $x^4$  con  $x$  che va da 1 a 10

$x^1$	$x^2$	$x^3$	$x^4$
1	1	1	1
2	4	8	16
3	9	27	81
...	...	...	...
10	100	1000	10000

# Esempio – Progetto dei cicli annidati

- Come si può stampare una tabella con righe e colonne?
  - Stampo l'intestazione (header)
    - Uso un ciclo for
  - Stampo il corpo della tabella
    - Quante righe ci sono nella tabella?
    - Quante colonne?
  - Un ciclo per le righe
    - Un ciclo interno per le colonne
- Nel nostro esempio ci sono:
  - 4 colonne
  - 10 righe

$x^1$	$x^2$	$x^3$	$x^4$
1	1	1	1
2	4	8	16
3	9	27	81
...	...	...	...
10	100	1000	10000

# Esempio – Pseudocodice (1)

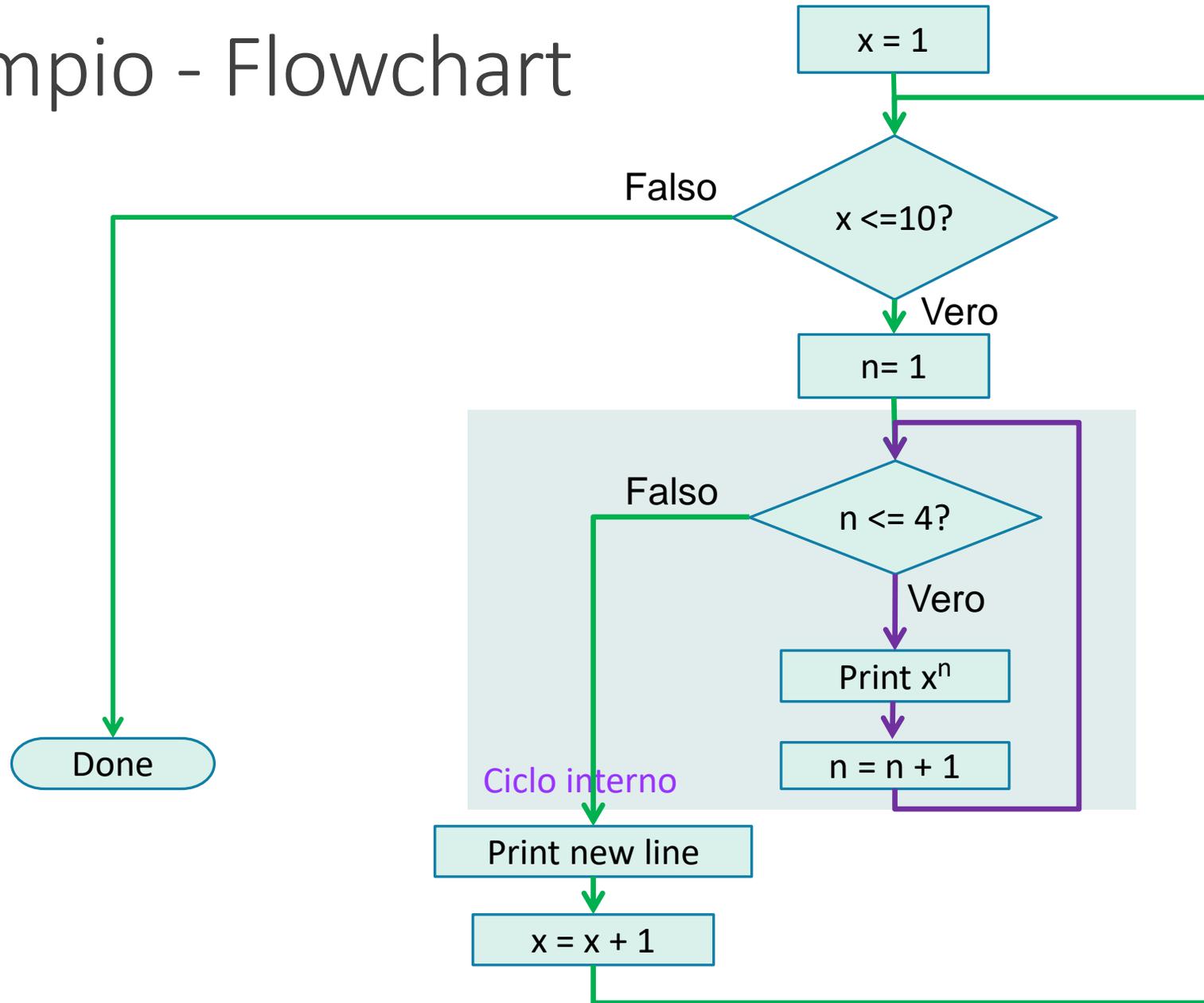
- Stampa l'header della tabella
  - for x da 1 a 10
    - stampa una nuova riga
    - stampa un carattere di «a capo»
  
- Come stampo una riga della tabella?
  - for n da 1 a 4
    - stampa  $x^n$
  
- Si deve mettere questo ciclo all'interno del precedente
  - Il ciclo interno è **annidato** all'interno di quello esterno

# Esempio – Pseudocodice (2)

```
stampa l'header della tabella
for x da 1 a 10
  for n da 1 a 4
    stampa  $x^n$ 
  stampa un «a capo»
```

	n →			
	$x^1$	$x^2$	$x^3$	$x^4$
x ↓	1	1	1	1
	2	4	8	16
	3	9	27	81
	...	...	...	...
	10	100	1000	10000

# Esempio - Flowchart



# Parametri opzionali della funzione `print`

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

Print *objects* to the text stream *file*, separated by *sep* and followed by *end*. *sep*, *end*, *file* and *flush*, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like `str()` does and written to the stream, separated by *sep* and followed by *end*. Both *sep* and *end* must be strings; they can also be `None`, which means to use the default values. If no *objects* are given, `print()` will just write *end*.

The *file* argument must be an object with a `write(string)` method; if it is not present or `None`, `sys.stdout` will be used. Since printed arguments are converted to text strings, `print()` cannot be used with binary mode file objects. For these, use `file.write(...)` instead.

Whether output is buffered is usually determined by *file*, but if the *flush* keyword argument is true, the stream is forcibly flushed.

<https://docs.python.org/3/library/functions.html#print>

<https://realpython.com/python-print/>

- `sep=`
  - Definisce il separatore da usare se si stampano diversi valori
  - Default: spazio ' '
  - `print(hour, min, sec, sep=':')`
- `end=`
  - Definisce cosa stampare alla fine di riga
  - Default: «a capo» '\n'
  - `print('Hello', end='')`
- `sep` e `end` sono «named parameters» (parametri nominati in maniera esplicita)
  - vedere Unità P5 per i dettagli...

# Stampare sulla stessa riga

- Possiamo usare il parametro opzionale `'end'`, per evitare che la `print` stampi un «a capo» dopo che gli argomenti sono visualizzati
- Si utilizza quando vogliamo stampare diversi elementi sulla stessa riga usando diverse istruzioni `print`.

- Per esempio le due istruzioni:

```
print("00", end="")  
print(3 + 4)
```

Produrranno come risultato:

```
007
```

- Includendo `end=""` come argomento, la funzione `print` stamperà una stringa vuota *dopo* gli argomenti, *al posto di un «a capo»*
- L'output della `print` successiva inizierà di conseguenza sulla stessa riga

# Powertable.py

```
1 #
2 # This program prints a table of powers of x.
3 #
4
5 # Initialize constant variables for the max ranges.
6 NMAX = 4
7 XMAX = 10
8
9 # Print table header.
10 #
11
12 for n in range(1, NMAX + 1) :
13     print("%10d" % n, end="")
14
15 print()
16 for n in range(1, NMAX + 1) :
17     print("%10s" % "x ", end="")
18
19 print("\n", "    ", "-" * 35)
20
21 # Print table body.
22 #
23
24 for x in range(1, XMAX + 1) :
25     # Print the x row in the table.
26     for n in range(1, NMAX + 1) :
27         print("%10.0f" % x ** n, end="")
28
29 print()
30
```

Il parametron **end=""** fa sì che l'”a capo” non sia stampato, e i numeri sono quindi stampati sulla stessa riga

Corpo del ciclo esterno, x = 1 → 10

Corpo del ciclo interno, n = 1 → 4

# Il risultato finale

```
[evaluate Powertable header.py]
```

```
      1      2      3      4  
      x      x      x      x
```

---

```
      1      1      1      1  
      2      4      8     16  
      3      9     27     81  
      4     16     64    256  
      5     25    125    625  
      6     36    216   1296  
      7     49    343   2401  
      8     64    512   4096  
      9     81    729   6561  
     10    100   1000  10000
```

# Esercizio

- Aprire il programma:
  - `powertable.py`
- Eseguire il programma ed osservare il risultato
- Implementare le seguenti modifiche:
  - Cambiare il valore di NMAX a 6 ed eseguire il programma
    - Cosa cambia nella tabella?
  - Ripristinare il valore di NMAX a 4
  - Cambiare il valore di XMAX a 4
    - Cosa cambia nella tabella?

# Esempi di cicli annidati (1)

Cicli annidati	Visualizzazione	Spiegazione
<pre>for i in range(3) :     for j in range(4) :         print("*", end="")     print()</pre>	<pre>**** **** ****</pre>	Visualizza 3 righe, ciascuna con 4 asterischi.
<pre>for i in range(4) :     for j in range(3) :         print("*", end="")     print()</pre>	<pre>*** *** *** ***</pre>	Visualizza 4 righe, ciascuna con 3 asterischi.
<pre>for i in range(4) :     for j in range(i + 1) :         print("*", end="")     print()</pre>	<pre>* ** *** ****</pre>	Visualizza 4 righe, di lunghezza crescente: 1, 2, 3 e 4.

# Hand-Tracing

- i assumerà i valori :
  - 0, 1, 2, 3 – quindi ci saranno quattro righe di asterischi
- j assumerà i valori
  - 0 – e ci sarà un asterisco
  - 0, 1 - e ci saranno due asterischi
  - 0, 1, 2 - e ci saranno tre asterischi
  - 0, 1, 2, 3- e ci saranno quattro asterischi

```
[evaluate nested loop example three.py]  
*  
**  
***  
****
```

```
1 for i in range(4) :  
2     for j in range(i + 1) :  
3         print("*", end="")  
4     print()
```

# Esempi di cicli annidati (2)

Cicli annidati	Visualizzazione	Spiegazione
<pre>for i in range(4) :     for j in range(5) :         if j % 2 == 1 :             print("*", end=" ")         else             print("-", end=" ")     print()</pre>	<pre>_*_*_ _*_*_ _*_*_ _*_*_</pre>	Visualizza trattini e asterischi alternati.
<pre>for i in range(3) :     for j in range(5) :         if i % 2 == j % 2 :             print("*", end=" ")         else             print(" ", end=" ")     print()</pre>	<pre>* * *  * * * * *</pre>	Visualizza uno schema a scacchiera.

# Secondo problema

- Stampare i seguente pattern di parentesi:

```
[ ] [ ] [ ] [ ]
```

```
[ ] [ ] [ ] [ ]
```

```
[ ] [ ] [ ] [ ]
```

- Il pattern è composto da :
  - Tre righe
  - Ogni riga ha quattro paia di parentesi
- Cosa sappiamo?
  - Abbiamo bisogno di 2 cicli annidati
    - Il primo ciclo stamperà ognuna delle 3 righe
    - Il secondo ciclo stamperà le 4 coppie di parentesi

# Pseudocode

```
For i = 0 to 2    # 3 rows
  For j = 0 to 3  # 4 cols
    Print “[ ]”
  Print a new line
```

```
1 for i in range(3) :
2   for j in range(4) :
3     print("[ ]", end="")
4   print()
```

```
[evaluate nested loop example three.py]
[ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ]
```

# Esercizio: Medie d'esame

---

# Definizione del problema

- Capita spesso di dover ripetutamente leggere ed elaborare diversi insiemi di valori:
  - Scrivere un programma che calcoli la media degli esami di un gruppo di studenti.
  - Ogni studente ha lo stesso numero di voti
  - Il programma deve chiedere in input il numero di esami
  - Quando si termina l'inserimento dei voti di uno studente, il programma deve chiedere se ci sono ulteriori studenti da processare
- Che dati abbiamo?
- Cosa dobbiamo calcolare?
- Quale algoritmo scegliere?

# Passo 1: capire il problema

- Per calcolare la media di uno studente, si devono acquisire e sommare tutti i voti di quello studente
  - Per fare questo si può usare un ciclo (vedere codice fatto in precedenza)
- Dobbiamo calcolare la media per diversi studenti (non solo uno)
  - Questo implica di usare cicli annidati
    - Un ciclo esterno per ogni studente
    - Un ciclo interno per processare i voti dello studente selezionato

# Passo 2

- Acquisire i voti di uno studente
- Preparare le variabili e impostare il ciclo
- È noto il numero di esami da processare, quindi si può impostare un ciclo controllato da un **contatore**.

*total\_score = 0*

*for i in range (1, number\_of\_exams + 1) :*

*Acquisire il prossimo voto*

*Sommare il voto al totale total\_score*

- Calcolare la media
- Stampare la media

# Passo 3

- Ripetere lo stesso processo per ogni studente
- Dato che non è noto il numero di studenti, dobbiamo utilizzare un ciclo **while** con **sentinella**
  - Per semplicità useremo “Y” come valore sentinella (Y=si, N=no)

# Passo 4: Python

```
1  ##
2  # This program computes the average exam grade for multiple students.
3  #
4
5  # Obtain the number of exam grades per student.
6  numExams = int(input("How many exam grades does each student have? "))
7
8  # Initialize moreGrades to a non-sentinel value.
9  moreGrades = "Y"
10
11 # Compute average exam grades until the user wants to stop.
12 while moreGrades == "Y" :
13
14     # Compute the average grade for one student.
15     print("Enter the exam grades.")
16     total = 0
17     for i in range(1, numExams + 1) :
18         score = int(input("Exam %d: " % i))    # Prompt for each exam grade.
19         total = total + score
20
21     average = total / numExams
22     print("The average is %.2f" % average)
23
24     # Prompt as to whether the user wants to enter grades for another student.
25     moreGrades = input("Enter exam grades for another student (Y/N)? ")
26     moreGrades = moreGrades.upper()
27
```

# Esempio - Medie esami

- Aprire il file:
  - examaverages.py
- Notare come il secondo ciclo (il ciclo for) è annidato nel ciclo **while**
  - Osserva e controlla l'indentazione

# Applicazione: Numeri casuali e simulazioni

---



4.9

# Numeri casuali/simulazioni

- I giochi spessissimo utilizzano numeri casuali per rendere le cose più interessanti
  - Lancio di dadi
  - Girare una ruota
  - Estrarre una carta
- Una simulazione generalmente richiede di ciclare attraverso una sequenza di eventi
  - Giorni / Minuti / Secondi
  - Eventi

# Generare numeri casuali

- Python mette a disposizione un *generatore di numeri casuali* che produce numeri che **sembrano** casuali
  - I numeri sono in realtà pseudo-casuali. Sono infatti estratti da una sequenza di numeri non consecutivi che non si ripetono (a meno di continuare nella sequenza per molto tempo)
  - `random()` ritorna un numero  $\geq 0$  e  $< 1$
- Il generatore di numeri casuali si trova nel modulo 'random'
  - `from random import random`
  - <https://docs.python.org/3/library/random.html>
  - <https://realpython.com/python-random/#prngs-in-python>

# Simulare un lancio di dadi

- Obiettivo:
  - Per generare un numero casuale intero in un certo intervallo (1-6) si utilizza la funzione `randint()`
  - `randint` ha due parametri che definiscono l'intervallo (inclusivo) dei numeri generati

## ch04/dice.py

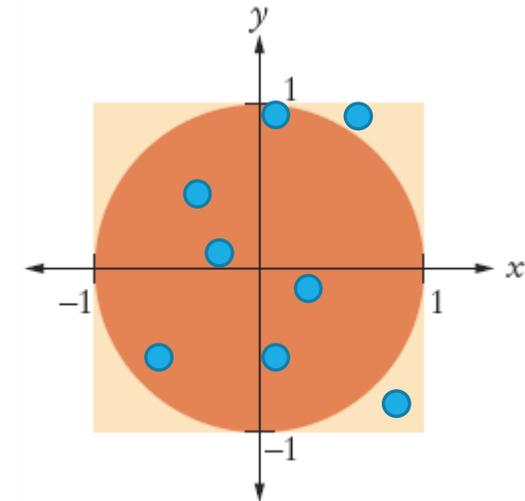
```
1  ##
2  # This program simulates tosses of a pair of dice.
3  #
4
5  from random import randint
6
7  for i in range(10) :
8      # Generate two random numbers between 1 and 6, inclusive.
9      d1 = randint(1, 6)
10     d2 = randint(1, 6)
11
12     # Print the two values.
13     print(d1, d2)
```

## Program Run

```
1 5
6 4
1 1
4 5
6 4
3 2
4 2
3 5
5 2
4 5
```

# Esempio - Il metodo Monte Carlo

- Utilizzato per trovare soluzioni approssimate a problemi che non possono essere risolti in modo semplice
- Esempio: Approssimare il valore di  $\pi$  usando l'area relativa di un cerchio all'interno di un quadrato
  - Utilizza semplice aritmetica
  - Si genera un numero casuale in  $[-1,1] \times [-1,1]$
  - Hits sono i numeri che cadono dentro al cerchio, i.e.  $x^2 + y^2 \leq 1$
  - Tries sono il numero di prove totali
  - Ratio è calcolato come  **$4 \times \text{Hits} / \text{Tries}$** 
    - N.B. 4 è l'area del quadrato



# Esempio - Il metodo Monte Carlo

```
1  ##
2  # This program computes an estimate of pi by simulating dart throws onto a square.
3  #
4
5  from random import random
6
7  TRIES = 10000
8
9  hits = 0
10 for i in range(TRIES) :
11
12     # Generate two random numbers between -1 and 1
13     r = random()
14     x = -1 + 2 * r
15     r = random()
16     y = -1 + 2 * r
17
18     # Check whether the point lies in the unit circle
19     if x * x + y * y <= 1 :
20         hits = hits + 1
21
22 # The ratio hits / tries is approximately the same as the ratio
23 # circle area / square area = pi / 4.
24
25 piEstimate = 4.0 * hits / TRIES
26 print("Estimate for pi:", piEstimate)
```

## Program Run

Estimate for pi: 3.1464

# Elaborazione di stringhe

---



4.8

# Elaborazione di stringhe

- Un tipico uso dei cicli è per l'analisi di stringhe.
- Ad esempio, si potrebbe voler
  - calcolare il numero di occorrenze di uno o più caratteri all'interno di una stringa o
  - verificare che il contenuto della stringa sia conforme a una qualche regola.

# Esempi di elaborazione di stringhe

- Contare i match
- Trovare tutti i match
- Trovare il primo e l'ultimo match
- Validare una stringa
- Costruire una nuova stringa

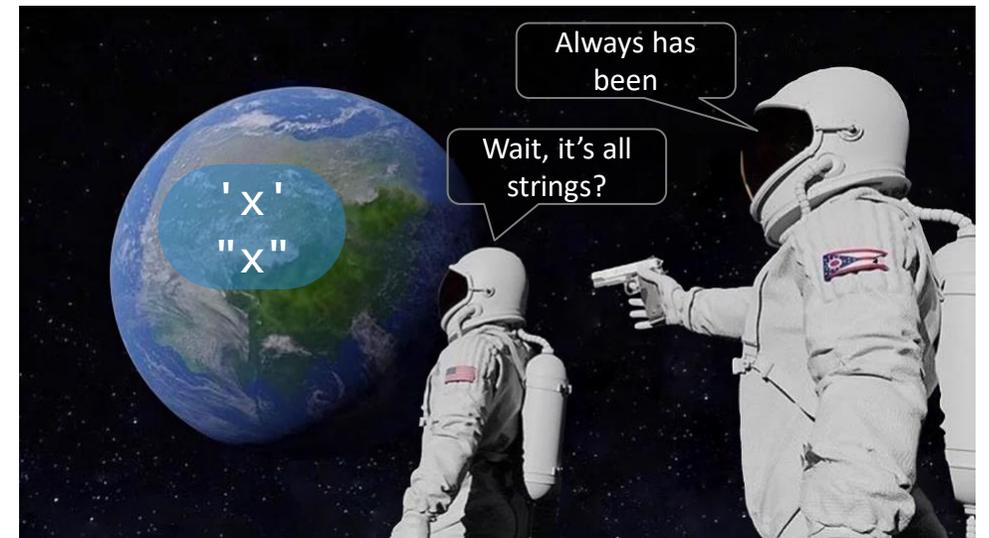
# Contare i match

- Si supponga di voler calcolare il *numero di lettere maiuscole* contenute in una stringa.
- Si può usare un ciclo **for** per controllare ogni carattere della stringa e verificare se sia maiuscolo
- Il ciclo sottostante setta la variabile **ch** via via a ogni carattere della stringa
- Ogni iterazione del ciclo controlla un carattere della stringa e verifica se sia maiuscolo

```
uppercase = 0
for ch in string :
    if ch.isupper() :
        uppercase = uppercase + 1
```

# Note

- for `ch` in `string` :
- A ogni iterazione, `ch` assume il valore del successivo carattere nella stringa
- In Python, `ch` è a sua volta una stringa (di lunghezza 1)
  - In Python non esiste il tipo “carattere” o “char”
  - È una grande differenza con C/C++/Java



# Contare vocali

- Si supponga di voler contare le **vocali** in una stringa
- Si può utilizzare un ciclo `for` per controllare se sia parte della stringa di sole vocali “`aeiou`”
- Il ciclo sottostante setta a ogni iterazione la variabile `ch` al prossimo carattere della stringa
- A ogni iterazione il ciclo verifica se `ch`, minuscolo, sia anche incluso nella stringa “`aeiou`” (volendo alla stringa si possono aggiungere le vocali accentate)

```
vowels = 0
for ch in word :
    if ch.lower() in "aeiou" :
        vowels = vowels + 1
```

# Ricerca di tutti i match

- Quando è richiesto di esaminare ogni carattere di una stringa, indipendentemente dalla sua posizione possiamo impostare un ciclo `for-in`
- Se **invece** dobbiamo stampare anche la **posizione** di ogni carattere **maiuscolo** in una frase, possiamo iterare sulle posizioni di ogni carattere (utilizzando un `for-range`) e usare la posizione per testare il carattere corrispondente
  - Se è maiuscolo, allora ne stampo la posizione
- Inizializziamo il **range** alla **length** (lunghezza) della stringa
  - Testiamo ogni carattere
  - Se maiuscolo, stampo la sua posizione **i** nella stringa

```
sentence = input("Enter a sentence: ")
for i in range(len(sentence)) :
    if sentence[i].isupper() :
        print(i) # the position, not the letter
```

# Ricerca del primo match

- Questo esempio trova la posizione della prima occorrenza di una cifra numerica in una stringa

```
found = False
position = 0
while not found and position < len(string) :
    if string[position].isdigit() :
        found = True
    else :
        position = position + 1

if found :
    print("First digit occurs at position", position)
else :
    print("The string does not contain a digit.")
```

# Ricerca dell'ultimo match

- Questo ciclo identifica la posizione dell'ultima cifra nella stringa
- Si utilizza un ciclo `while` che parte dall'**ultimo** carattere nella stringa e testa ogni carattere successivo muovendosi verso l'inizio della stringa
  - **Position** è inizializzata alla lunghezza della stringa - 1
  - Se il carattere non è una cifra, si decrementa **position** di 1
  - Si continua fino a che si trova una cifra o non ci sono più caratteri da analizzare

```
found = False
position = len(string) - 1
while not found and position >= 0 :
    if string[position].isdigit() :
        found = True
    else :
        position = position - 1
```

# Validare una Stringa

- Negli Stati Uniti, i numeri di telefono sono fatti di 3 parti: area code exchange, and line number, solitamente scritti nella forma:

(###)###-####

# Validare una Stringa (codice)

- È possibile esaminare una stringa per verificare che contenga un numero di telefono formattato correttamente (es.: (703)321-6753)
- Il ciclo dovrà testare ogni carattere per verificare che sia corretto

```
valid = len(string) == 13 # set valid to True or False
position = 0
while valid and position < len(string) :
    valid = ((position == 0 and string[position] == "(")
            or (position == 4 and string[position] == ")")
            or (position == 8 and string[position] == "-")
            or (position != 0 and position != 4 and position != 8
                and string[position].isdigit()))
    position = position + 1
```

# Costruire una nuova stringa

- Una scomodità comune nei siti di acquisti online è che richiedono di inserire i numeri di CC o IBAN senza spazi o trattini, il che rende la verifica (visuale) del numero inserito piuttosto scomoda.
- Quanto può essere difficile rimuovere spazi e trattini da una stringa?

**Credit Card Information** *(all fields are required)*

We Accept:   

Credit Card Type:

Credit Card Number:

*(Do not enter spaces or dashes.)*

# Costruire una nuova stringa (codice)

- **Il contenuto di una stringa non si può modificare**
  - Le stringhe in Python sono immutabili
  - Nulla ci vieta di creare una **nuova stringa**.
- Questo ciclo costruisce una nuova stringa contenente un numero di carta di credito senza spazi o trattini :
  - Si legge il numero di carta di credito
  - Si inizializza la nuova stringa come vuota (= `""`)
  - Si testa ogni carattere inserito dall'utente
    - Se il carattere non è uno spazio e non è un trattino, si appende alla nuova stringa (+ `char`)

```
userInput = input("Enter a credit card number: ")
creditCardNumber = ""
for char in userInput :
    if char != " " and char != "-" :
        creditCardNumber = creditCardNumber + char
```

# Esempio: BlackJack

---

PROBLEM SOLVING

# BlackJack

- Realizzare un algoritmo e scrivere un programma in Python che giochi a BlackJack contro l'utente
- Si assuma che la mano si svolga tra il banco e un solo giocatore
- L'obiettivo del BlackJack è di avere in mano delle carte la cui somma sia il più possibile vicina a 21, ma senza superarlo
  - È necessario tenere traccia della somma delle carte che i due giocatori (banco e giocatore) ricevono durante la mano

# Regole del gioco (semplificate)

- Fase 1: il computer (banco) si dà una carta e la fa vedere al giocatore
- Fase 2: il computer dà due carte al giocatore
- Fase 3: in base alla somma delle due carte ricevute, il giocatore sceglie se ricevere un'altra carta (hit) o fermarsi (stay)
  - Questa fase si ripete fino a che il giocatore si ferma (stay) o la somma delle sue carte è  $>21$ , nel qual caso il giocatore ha perso
- Fase 4: se il giocatore non ha perso, il computer gioca. In base alla somma delle sue carte:
  - Se la somma è  $< 17$  → il computer **deve prendere** una nuova carta
  - Se la somma è  $\geq 17$  → il computer **deve fermarsi**
  - Se la somma è  $> 21$  → il computer **perde**

# Fine del gioco

- Se il banco supera 21, il giocatore vince
- Se entrambi si sono fermati, allora le due somme si confrontano e vince chi ha la somma maggiore. In caso di parità vince il banco.

# Semplificazioni

- Si estraggono le carte da un mazzo 'infinito'
  - A ogni estrazione, tutte le carte hanno la stessa probabilità di essere estratte
- Le carte hanno un valore da 1 a 10
  - Sono tutti interi
  - L'asso vale 1
  - Non preoccuparsi delle figure
  - Non preoccuparsi del seme ♠ ♥ ♦ ♣

[https://en.wikipedia.org/wiki/Playing\\_cards\\_in\\_Unicode](https://en.wikipedia.org/wiki/Playing_cards_in_Unicode)

# Approfondimenti (per casa)

- Estrazione delle carte:
  - Si immagini di estrarre le carte da  $N$  mazzi. In questo caso quando si estrae una carta, le probabilità dipendono da quali carte sono state estratte in precedenza
- Valore dell'asso:
  - Quando si estrae un asso, il suo valore può essere 1 o 11.
- Introdurre la possibilità di far giocare più di un giocatore (oltre al banco)

[https://en.wikipedia.org/wiki/Playing\\_cards\\_in\\_Unicode](https://en.wikipedia.org/wiki/Playing_cards_in_Unicode)

# Sommario

---

# Sommario: due tipi di ciclo

- Cicli **while**
- Cicli **for**
- I cicli **while** sono in generale più versatili
- Si usano i cicli **for** per:
  - Iterare sugli elementi contenuti in un **contenitore**
  - Creare un ciclo controllato da un **contatore** di interi
- La funzione **enumerate** permette di ciclare contemporaneamente sugli indici e sui valori di un contenitore
- Per interrompere anticipatamente una singola iterazione, usare **continue**
- Per interrompere anticipatamente tutto il ciclo, usare **break**

# Sommario

- Ogni ciclo richiede le seguenti parti:
  - **Inizializzazione** (preparare le variabili per iniziare il ciclo)
  - **Condizione** (per capire se, ad ogni iterazione, il ciclo deve essere eseguito o no)
  - **Aggiornamento** (modificare qualche variabile all'interno del ciclo, in modo che prima o poi le iterazioni terminino)
  - Un ciclo **for-range** lo fa automaticamente, mentre in un ciclo **while** è il programmatore che deve aggiornare le variabili
- Un ciclo esegue il suo blocco di istruzioni fintanto che la condizione è Vera.
- Sbagliare il numero di ripetizioni di 1 è un errore molto comune.
  - Ragionare su cosa succede nei casi più semplici aiuta ad evitare questo errore

# Sommario

- Un valore **sentinella** serve ad identificare la fine di un insieme di dati, ma non fa mai parte dei dati stessi
- È possibile utilizzare una variabile booleana per controllare un ciclo
  - Si setta la variabile a **True** prima di entrare nel ciclo
  - Si setta la variabile a **False** per uscire dal ciclo
- I cicli sono utilizzati in molti problemi di manipolazione ed analisi delle stringhe
- In una simulazione, si usa un computer per simulare un'attività
  - Si può introdurre il concetto di «casualità» utilizzando i generatori di numeri casuali.