PYTHON
LA REPL DI PYTHON
VARIABILI E TIPI DI DATO
COSTRUTTI DI SELEZIONE
CICLI WHILE
CICLI FOR

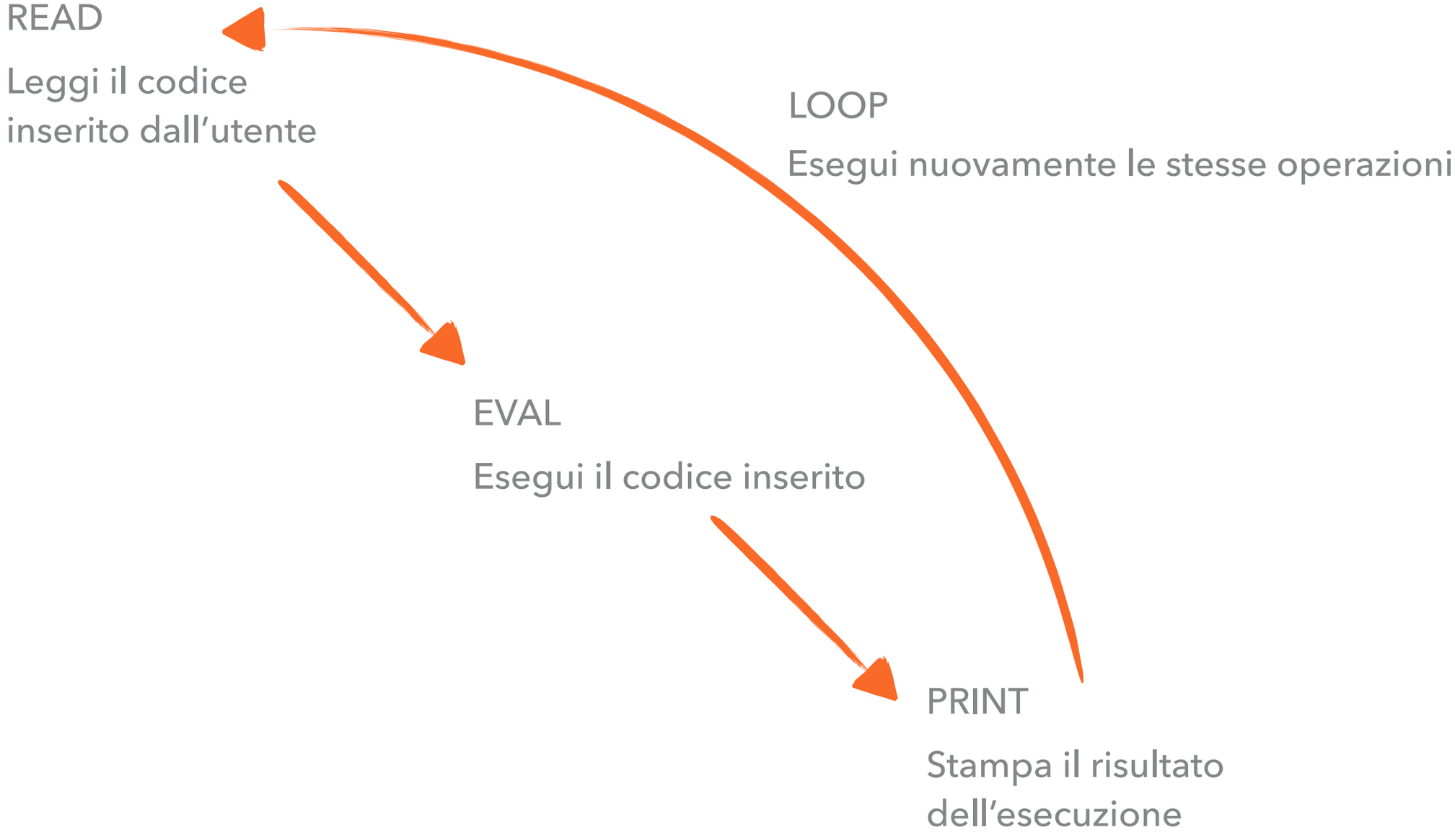
INFORMATICA

PYTHON: LA STORIA

- “Python is an interpreted, interactive, object-oriented programming language. “
- Creato nel 1991 da Guido Van Rossum
- Versione 3 rilasciata nel 2008
- Oggi un linguaggio molto utilizzato, soprattutto nell’ambito della data science, del machine learning e dell’intelligenza artificiale

I Monty Python, da cui deriva il nome Python

READ-EVAL-PRINT-LOOP



TIPI DI DATO

- Il tipo di un dato ci dice come interpretare un'area di memoria...
- ...e che operazioni possiamo effettuare con quel dato
- Molteplici tipi numerici: interi, floating point, complessi
- Booleani (rappresentano solo vero e falso)
- Stringhe (sequenze di caratteri)

TIPI DI DATO NUMERICI

- **int**: numeri interi (positivi e negativi)
- **float**: numeri a virgola mobile, *solitamente* a precisione doppia (64 bit)
- **complex**: numeri complessi
- Questi sono solo i tipi più comuni, ne esistono anche altri utilizzabili (e.g., per rappresentare frazioni o per usare una differente precisione)

OPERAZIONI SUI TIPI NUMERICI

- somma: **x + y**, sottrazione: **x - y**, moltiplicazione: **x * y**
- divisione: **x / y** e divisione intera: **x // y** (tiene solo la parte intera del quoziente)
- modulo: **x % y**, restituisce il resto della divisione (solo tra interi)
- elevamento a potenza: **x ** y**
- possiamo sempre cambiare la priorità delle operazioni tramite parentesi tonde (e *solo* parentesi tonde)

BOOLEANI

- È un tipo di dato con solo due valori possibili:
- **True**, per indicare che qualcosa è vero
- **False**, per indicare che qualcosa è falso
- Vengono utilizzati per verificare condizioni
- Sono il risultato di comparazioni tra valori di altri tipi

OPERAZIONI SUI BOOLEANI

- Possiamo combinare i valori Booleani con le operazioni di **and**, **or** e **not**.

A	not A
False	True
True	False

A	B	A and B	A or B
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

OPERAZIONI CHE RESTITUISCONO BOOLEANI

- **x < y**. True se x è *minore* di y
- **x > y**. True se x è *maggiore* di y
- **x <= x**. True se x è *minore o uguale* a y
- **x >= x**. True se x è *maggiore o uguale* a y
- **x == y**. True se x e y sono *uguali*
- **x != y**. True se x e y sono *diversi*

STRINGHE

- Una stringa rappresenta una sequenza di caratteri
- Quando la scriviamo, una stringa è una sequenza di testo racchiusa da apici singoli o doppi

- Notare la differenza:
  - 3 è il numero 3.
  - “3” è la sequenza di caratteri che, stampata a schermo, fa apparire il carattere “3”.

OPERAZIONI SUI STRINGHE

- **“ciao” + “mondo”** produce la stringa “ciaomondo”. Se volessimo uno spazio dovremmo inserirlo:
  “ciao ” + “mondo” produce “ciao mondo”

- \+ tra stringhe è **concatenazione**
  \+ tra numeri è **addizione**.
  Non sono intercambiabili

- Alcuni simboli nelle stringhe sono interpretati in modo diverso: “\n” rappresenta il ritorno a capo

CONVERSIONE TRA TIPI

- Come facciamo a passare dalla stringa che contiene “3” al numero “3”? Ci sono funzione per passare tra tipi diversi
- **int(“3”)** restituisce il numero 3
- **float(“3.14”)** restituisce il numero 3.14
- **str(4)** restituisce la stringa “4”
- Se una conversione non è possibile viene generata una eccezione (simile ad un “errore”)

PERCHÉ USARE LE VARIABILI?

- Non vogliamo lavorare direttamente con degli indirizzi di memoria
- Vogliamo dare un nome a delle celle di memoria
  e.g., “massa” invece di “cella numero 0895”
- Se ci serve più di una cella vogliamo usare un solo nome per più celle consecutive e dire come vogliamo interpretare il loro contenuto

ASSEGNAMENTO: STRUTTURA DI BASE

Nome di una variabile, se il nome esiste già
il valore precedentemente contenuto sarà sovrascritto

```
nome_della_variabile = espressione
```

Una espressione può essere:

- Una costante
- Una variabile
  in questo caso usiamo il *valore* della variabile nell'espressione
- Una combinazione di altre espressioni con somma, prodotto, etc.
  (a seconda di cosa è consentito dal tipo di dato)
- Possiamo usare parentesi tonde per raggruppare e dare priorità

ESEMPI DI ASSEGNAMENTO

`base = 3`

int: valore intero
e.g., 3, 5, 6, 12, -987

`temperatura = 3.14`

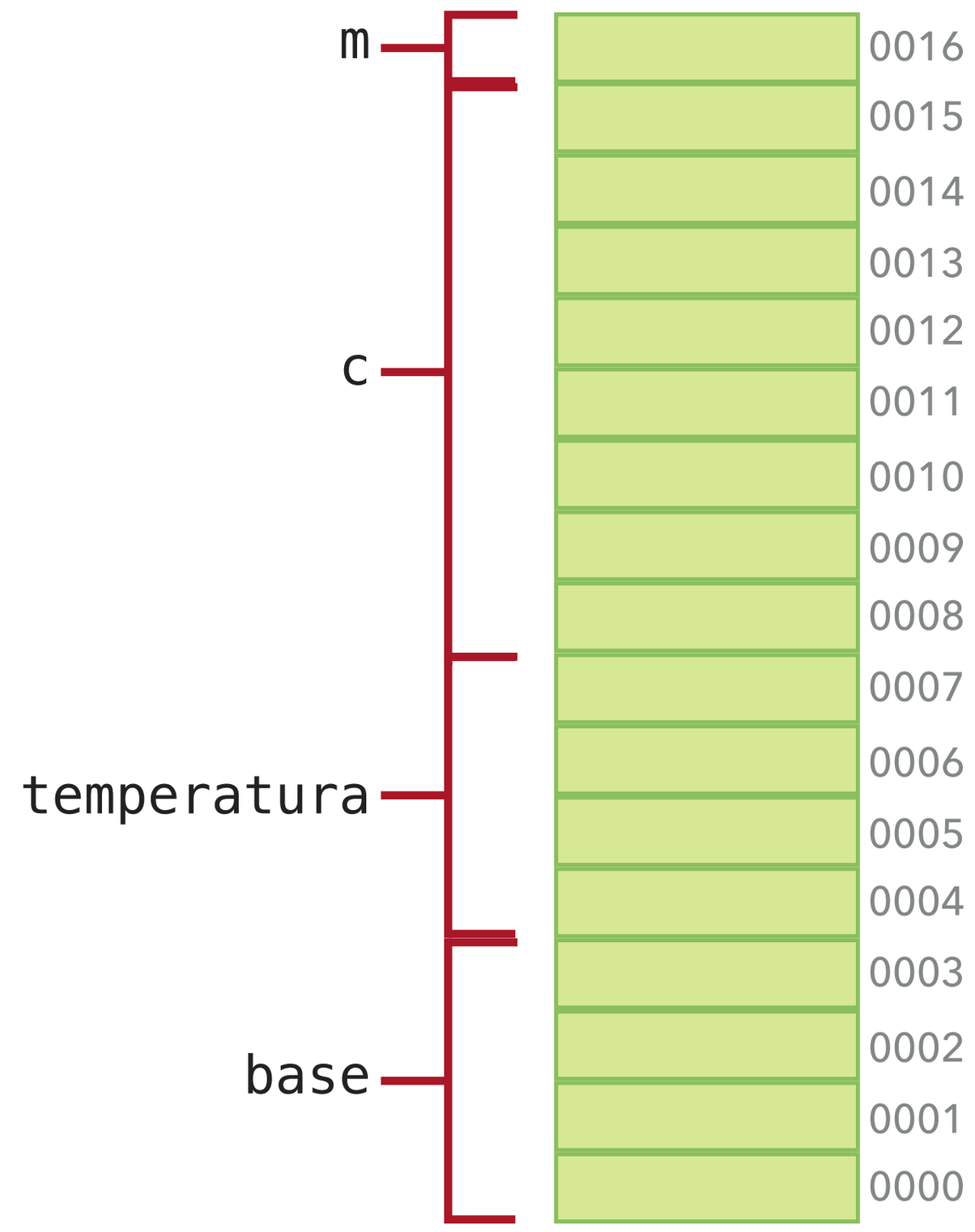
float: valore con la virgola
e.g., 4.5, 3.0, -2.3

`c = 4-2j`

complex: numero complesso
e.g., 4.5 + 5j, 1j, 3 + 0.5j

`m = True`

Booleano: uno solo tra True e False



ATTENZIONE!

ASSEGNAMENTO NON È UGUAGLIANZA

```
nome_della_variabile = espressione
```

LATO SINISTRO

UNA POSIZIONE IN MEMORIA
DOVE SALVARE UN VALORE

LATO DESTRO

QUALCOSA CHE PUÒ VENIRE VALUTATO
E CHE RITORNA UN VALORE

L'operatore è asimmetrico:
lato sinistro e destro non sono invertibili

OUTPUT SU SCHERMO

- Spesso vogliamo scrivere qualcosa all'utente del programma
- Per questo usiamo la funzione print
- Print prende come argomento qualcosa che può essere convertito in una stringa e lo visualizza
- esempi: **print(3)**, **print("hello")**, **print("x = " + str(3))**

INPUT DA TASTIERA

- Per leggere quello che scrive l'utente possiamo utilizzare la funzione input.
- Input attende che l'utente scriva qualcosa e prema invio, quello che è stato scritto dall'utente viene restituito come una stringa.
- ```
  >>> x = input()
  4
  >>> int(x) + 4
  8
  ```
  ```

COSA COMMENTARE

- Commentare all'inizio del programma per spiegare cosa deve fare:
  scoprire se un numero è primo,
  simulare un fenomeno fisico,
  conquistare il mondo.

- Evitare commenti ovvi:
  x = x + 1 # incremento x di uno

- Spiegare perché si fa qualcosa, soprattutto se non è ovvio

COMMENTI

E se volessimo aggiungere delle note nel codice?

```
temperatura = 12
```

Vogliamo dire da che usiamo Celsius per questo valore...

```
temperatura = 12 # Valore in Celsius
```

Da # a fine linea l'interprete ignora cosa viene scritto

TIPI DI COMMENTI

```
x = 4 # Commento su una singola linea


# Commento su linee multiple.
# Anche questo viene ignorato
# Fino a qui


"""
Commento su linee multiple.
Anche questo viene ignorato se usiamo
dei tre volte dei doppi apici
Fino a qui.
"""
```

SCELTA CON "IF"

- Abbiamo visto come eseguire operazioni su diversi tipi di dato
- Abbiamo visto come dare un nome ad aree di memoria con le variabili
- Possiamo compiere operazioni durante l'assegnamento
- Ma **non** possiamo ancora cambiare quale sarà la successiva operazione da eseguire!
- Per questo useremo l'operatore di scelta "if"

SCELTA: FORMA DI BASE DEL COMANDO "IF"

Qualcosa che ritorna "vero" o "falso".

Ad esempio:
x < 4
x > y

```
if condizione:
    Codice da eseguire se la condizione è vera
```

Può essere un qualsiasi insieme di istruzioni
incluso, se necessario, altri "if"

L'indentazione è importante,
serve a capire quali parti di codice
devo essere eseguire in modo condizionale!

CONFRONTI

a<b	Minore
a>b	Maggiore
a<=b	Minore o uguale
a>=b	Maggiore o uguale
a==b	Uguale
a!=b	Diverso

ATTENZIONE

== e = sono diversi
e non devono essere confusi.

```
x = 2 # assegna ad x il valore 2
x == 2 # vero se x vale 2
```

ESEMPI DI IF

```
x = 3
if x > 2:
  x = 4
```

Questa istruzione viene eseguita

```
x = 3
if x < 2:
  x = 4
```

Questa istruzione **non** viene eseguita

QUIZ SU "IF"

```
x = 3
y = 2*x
x = 5
if y == 6:
  x = 7
```

Quali sono i valori di x e y dopo l'esecuzione di questo codice?

1. x è 5
   y è 10
2. x è 7
   y è 6
3. x è 7
   y è 14
4. x è "Darth Vader"
   y è "Duca Leto Atreides"

SCELTA: FORMA CON ELSE DEL COMANDO "IF"

Forma più generica del comando "if"

```
if condizione:
   Codice da eseguire se la condizione è vera
else:
   Codice da eseguire se la condizione è falsa
```

Uno solo dei due pezzi di codice viene eseguito.

Quale dei due dipende dalla condizione

SCELTA: FORMA CON ELIF DEL COMANDO “IF”

Forma utilizzata per evitare di innestare troppi “if”:

```
if condizione1:
   Codice da eseguire se la condizione1 è vera
elif condizione2:
   Codice da eseguire se la condizione1 è falsa
   e la condizione2 è vera
else:
   Codice da eseguire se entrambe le precedenti
   condizioni sono false (la parte “else” è facoltativa)
```

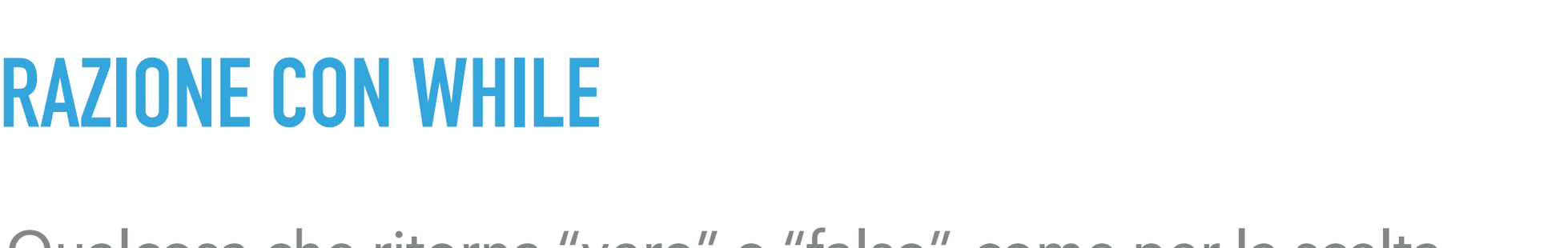
Uno solo dei tre (resp., degli n) pezzi di codice viene eseguito.

Quale dei tre (resp., degli n) dipende dalle due (resp., $n - 1$) condizioni

ITERAZIONE CON WHILE

Qualcosa che ritorna “vero” o “falso”, come per la scelta

```
while condizione:
    codice da eseguire se la condizione è vera.
    Il codice viene eseguito di nuovo se la condizione
    rimane vera
```



```
if condizione:
    codice da eseguire se la condizione è vera.
    Torna ad eseguire nel punto indicato dalla freccia.
```

ESEMPI DI WHILE

```
x = 0
y = 0
n = 0
while x < 10:
  y = 0
  while y < 10:
    y = y + 1
    n = n + 1
  x = x + 1
```

Il ciclo esterno viene eseguito 10 volte

Il ciclo interno viene eseguito 10 volte...
... ad ogni esecuzione del ciclo esterno

Quindi n viene incrementato 100 volte!

QUIZ SU "WHILE"

```
x = 0
y = 1
while x < 4:
   x = x + 1
   y = 2 * y
```

Quali sono i valori di x e y dopo l'esecuzione di questo codice?

1. x è 5
   y è 8
2. Ciclo
   infinito
3. x è 4
   y è 16
4. x è "Il capitano Kirk"
   y è "gli ABBA"

I CICLI FOR

STRUTTURA COMUNE DELL'ITERAZIONE

Variabile "indice"

Dichiariamo ed inizializziamo una variabile che conta
il numero di volte che abbiamo eseguito il ciclo

Condizione di uscita

Controlliamo se proseguire il ciclo

```
i = 0
while i < 10:
    # qui mettiamo il nostro codice
    i = i + 1
```

Incremento della variabile indice

Aumentiamo il valore della variabile indice

IDEA: NON RIPETERE INUTILMENTE

- La precedente struttura di iterazione è molto comune
- Ha senso riscriverla ogni volta rischiando di sbagliare?
- **NO**
- Per questo il linguaggio Python fornisce un costrutto pensato apposta per questo: il **ciclo for**

STRUTTURA DEL CICLO FOR

Variabile “indice”

La definiamo all’inizio del ciclo for, assumere tutti i valori della “collezione” specificata dopo “in”.

Collezione

Una sequenza di valori da assegnare, uno per ciclo, alla variabile indice

```
for i in range(0,10):
    # qui mettiamo il nostro codice
```

Cosa è “Range”?

Range(inizio,fine) ci permette di generare una sequenza di numeri nell’intervallo [inizio, fine) a distanza 1 l’uno dall’altro

EQUIVALENZA

```
i = 0
while i < 10:
    # qui mettiamo il nostro codice
    i = i + 1
```

```
for i in range(0,10):
    # qui mettiamo il nostro codice
```

Questo funziona perché range crea una collezione di numeri che partono da 0 e sono incrementi di 1 fino a che sono minori di 10

ESEMPIO: CICLI ANNIDATI

Ciclo esterno, viene eseguito 3 volte

```
for i in range(1,4):
  for j in range(1,4):
    prodotto = i * j
    print(str(i) + " * " + str(j)
          + " = " + str(prodotto))
```

Ciclo interno, viene eseguito 3 volte per ogni esecuzione del ciclo esterno, quindi 9 volte in totale

Output:

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
```

QUIZ SU “FOR”

```
x = 5
for i in range(0,5):
    print(x * i)
```

Quale è l’output di questo codice?

1. 0 1 2 3 4
2. 5 5 5 5 5
3. 0
4. 0 5 10 15 20

QUIZ SU "FOR"

```
z = 10
for x in range(0,z):
    for y in range (0,z):
        print("x =" + str(x))
```

Quale è l'output di questo codice?

1. x = 0
   x = 1
   ...

2. x = 1
   x = 2
   ...

3. x = 0
   x = 0
   ...

4. Never gonna give you up
   Never gonna let you down
   ...