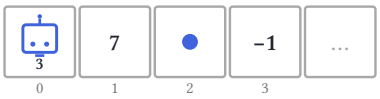
Dal robot al codice

Dalla logica del robot al computer e al primo programma in Python



`avanza()` `gira(destra)` `ripeti` `se` `leggi(x)` `if` `while` `print()`

★☆☆ facile ★★☆ medio ★★★ difficile

Indice

1 Le mosse del robot **3**
- 1.1 Il mondo del robot 3
- 1.2 Muoversi 3
- 1.3 Ricordare e comunicare 3
- 1.4 Decidere e ripetere 4
- 1.5 I problemi 4

2 Come funziona un computer **10**
- 2.1 Dentro la scatola 10
- 2.2 Il bit: l'alfabeto del computer 10

3 Introduzione a Python **13**
- 3.1 Variabili 13
- 3.2 Liste 14
- 3.3 Decidere: if ed else 14
- 3.4 Ripetere: while e break 15
- 3.5 Parlare con l'utente: print 15
- 3.6 Dal robot a Python: cosa cambia 16
- 3.7 Esercizi introduttivi 16
- 3.8 Esercizi più avanzati 18

1 Le mosse del robot

Programmare è spiegare a qualcuno, passo per passo, come fare una cosa — senza lasciare niente al caso. Il nostro qualcuno è un piccolo robot: non pensa, ma esegue con precisione assoluta le istruzioni che gli diamo. In questa prima sezione impariamo tutto il suo vocabolario; lo metteremo poi alla prova con dieci problemi.

1.1 Il mondo del robot

Il robot vive in una griglia di caselle e guarda sempre in una direzione: destra, sinistra, sopra o sotto. Le liste su cui viaggia possono essere così lunghe che nessuno sa dove finiscono. Ogni casella può essere di tre tipi:

- **Vuota** — ci si può camminare sopra senza che succeda niente.
- **Con un oggetto** — appena il robot entra nella casella, l'oggetto viene raccolto automaticamente.
- **Con un numero nascosto** — lo si può leggere con l'istruzione `leggi()`, che conosceremo tra poco.



Il robot è sulla casella 0, che nasconde il numero 3. La casella 2 nasconde un 7, la casella 3 contiene un oggetto, e la lista continua all'infinito.

1.2 Muoversi

Per muoversi il robot conosce solo due gesti. Tutto il resto — camminate, dietrofront, percorsi a zigzag — è una combinazione di questi.

`avanza()` **Avanza**
il robot si sposta di una casella nella direzione in cui sta guardando.

`gira(destra | sinistra)` **Gira**
il robot ruota su se stesso di 90°, in senso orario (destra) o antiorario (sinistra), senza cambiare casella.

Per fare dietrofront lungo una striscia servono sempre due girate di fila: uno per guardare dalla parte opposta, l'altro per... no, davvero, ne bastano due.

1.3 Ricordare e comunicare

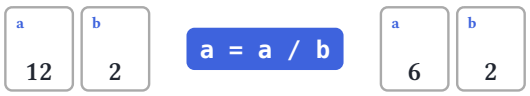
Nella pancia il robot porta una memoria fatta di caselle etichettate con le lettere dell'alfabeto. Per ora ne ha due: **a** e **b**. All'inizio ogni casella della memoria contiene 0.

`leggi()` **Leggi**
restituisce il numero nascosto nella casella su cui si trova il robot in quel momento.

`leggi(x)` **Memorizza**
deposita nella casella x della memoria il numero della casella attuale. Attenzione: il numero che c'era prima è perso per sempre!

`scrivi(x)` **Comunica**
aggiunge in fondo allo scontrino il numero contenuto nella casella x della memoria. Lo scontrino è il modo che il robot ha di comunicare con noi.

La memoria si può anche aggiornare con i calcoli: ogni istruzione deposita in una casella un valore calcolato con +, −, × e ÷ a partire dai valori attuali — e anche da `leggi()`, che porta il numero della casella su cui il robot si trova. Per esempio, partendo con **a** = 12 e **b** = 2:



*La memoria prima e dopo: **a** diventa 6, mentre **b** resta 2.*

1.4 Decidere e ripetere

Finora il robot ha sempre eseguito tutto quello che gli scrivevamo, nell'ordine esatto. Con due nuove istruzioni impara a scegliere e a ripetere da solo. Le istruzioni «dentro» un **se** o un **ripeti** si scrivono rientrate di due spazi, come negli esempi qui sotto.

`se … altrimenti …` **Decidere**
esegue le istruzioni rientrate solo se la condizione è vera; con altrimenti, ne esegue altre quando è falsa. Le condizioni confrontano numeri con =, ≠, <, >, ≤ e ≥.

`ripeti: … stop` **Ripetere**
ripete per sempre le istruzioni rientrate. L'istruzione stop lo interrompe: il robot prosegue dalla prima istruzione che viene dopo il ripeti.

```
se a < b:
  avanza()
altrimenti:
  scrivi(b)
```

Se a è minore di b il robot avanza; altrimenti scrive il valore di b sullo scontrino.

```
ripeti:
  avanza()
  se leggi() = 0:
    stop
```

Il robot avanza finché non trova uno 0: allora stop lo fa uscire dal ripeti.

1.5 I problemi

Dieci problemi, più o meno in ordine di difficoltà: le stelle in alto a destra indicano quanto. Ogni problema ha sempre la stessa forma: uno **scenario** che descrive il mondo, un **obiettivo** da raggiungere e un **esempio** di input e output. Tre convenzioni valgono per tutti: il robot parte sempre dalla prima casella; le posizioni si contano partendo da 0 (la casella di partenza è la posizione 0); le liste con i puntini continuano all'infinito. Tutte le istruzioni sono quelle imparate qui sopra — e se servirà una memoria più capace, verrà annunciato.

Problema 1 — Raccogli gli oggetti

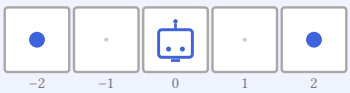
★☆☆

Scenario. Il robot parte dalla casella 0 di una striscia di caselle, rivolto verso destra. Due oggetti lo aspettano: uno sulla casella −2 e uno sulla casella +2. Quando il robot entra in una casella con un oggetto, lo raccoglie automaticamente.

Obiettivo. — Raccogli entrambi gli oggetti con una sola sequenza di istruzioni.

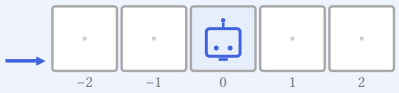
ESEMPIO

INPUT



il robot parte dalla casella 0, rivolto verso destra

OUTPUT



una possibile conclusione: oggetti raccolti 2 su 2, e il robot è tornato alla partenza

Problema 2 — La passeggiata

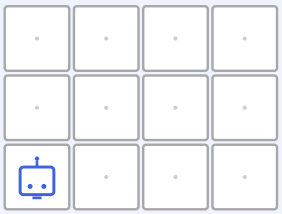
★★☆

Scenario. Il robot si trova nell'angolo in basso a sinistra di una griglia 4×3 (quattro colonne e tre righe), rivolto verso destra. Le caselle sono tutte vuote.

Obiettivo. — Fai una passeggiata che attraversi tutte le caselle della griglia almeno una volta.

ESEMPIO

INPUT



il robot parte dall'angolo in basso a sinistra, rivolto verso destra

OUTPUT



una delle tante passeggiate possibili: 12 caselle su 12, nell'ordine indicato dai numeri

Problema 3 — La somma della griglia

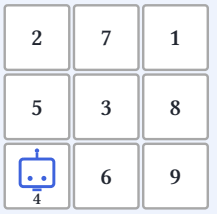
★★☆

Scenario. Come nella passeggiata, ma questa volta ogni casella della griglia nasconde un numero. Il robot parte dall'angolo in basso a sinistra.

Obiettivo. — Trova la somma di tutti i numeri della griglia e falla comparire in fondo allo scontrino.

ESEMPIO

INPUT



ogni casella nasconde un numero; il robot parte da quella con il 4

OUTPUT



2 + 7 + 1 + 5 + 3 + 8 + 4 + 6 + 9 = 45

Problema 4 — Fermati al −1

★☆☆

Scenario. Il robot viaggia su una lista di caselle indefinitamente lunga: nessuno sa quante ce ne sono. Ogni casella nasconde un numero, e da qualche parte, prima o poi, c'è una casella che nasconde −1.

Obiettivo. — Avanza fino alla prima casella con il numero −1 e fermati lì.

ESEMPIO

INPUT



la lista continua all'infinito: da qualche parte c'è un −1

OUTPUT



il robot si ferma sulla casella con il −1 (posizione 3)

Problema 5 — Conta i passi

★★☆

Scenario. Come nel problema precedente: lista infinita, numeri nascosti, e da qualche parte un −1.

Obiettivo. — Fermati al −1 e scrivi sullo scontrino quanti passi hai fatto per arrivarci, cioè quante caselle lo separano dalla partenza.

ESEMPIO

INPUT



quanto è lontano il −1?

OUTPUT

SCONTRINO

4

il robot si ferma sul −1 e scrive: 4 passi dalla partenza

Problema 6 — La somma del viaggio

★★☆

Scenario. Ancora la solita lista infinita di numeri, con un −1 a chiuderla.

Obiettivo. — Avanza fino al −1 sommando man mano tutti i numeri incontrati lungo il viaggio (il −1 non si somma!). Poi scrivi la somma sullo scontrino.

ESEMPIO

INPUT



somma tutto quello che incontri lungo il cammino

OUTPUT

SCONTRINO

8

3 + 1 + 4 = 8, poi il robot si ferma sul −1

Problema 7 — La corsa crescente

Scenario. Lista infinita di numeri, ma stavolta nessun −1: è il ritmo dei numeri a decidere quando fermarsi.

Obiettivo. — Avanza finché ogni numero è più grande del precedente; fermati appena la crescita si interrompe, sull'ultimo numero crescente.

ESEMPIO

INPUT



i numeri crescono... finché crescono

OUTPUT



il 4 è più piccolo del 7: il robot si ferma sul 7 (posizione 2)

Problema 8 — Dove si ferma la corsa

Scenario. Come la corsa crescente, ma con una richiesta in più.

Obiettivo. — Fermati quando i numeri smettono di crescere e scrivi sullo scontrino la posizione raggiunta. Ricorda: la casella di partenza è la posizione 0.

ESEMPIO

INPUT



come prima, ma ora ci vuole anche la posizione

OUTPUT



il robot si ferma sul 7, che si trova sulla casella 2

Problema 9 — Serie su serie

★★★

Scenario. Questa volta la lista è organizzata in serie: prima una serie di numeri positivi, poi un −1 che la chiude, poi la serie successiva… e quando tutte le serie sono finite, un −2 a salutare.

4	1	7	2	−1	4	5	−1	5	7	5	7	3	−2

Tre serie: (4, 1, 7, 2), (4, 5) e (5, 7, 5, 7, 3).

Obiettivo. — Calcola la somma di ciascuna serie e scrivi sullo scontrino la media delle somme.

ESEMPIO

INPUT



tre serie: (2, 4), (4, 5) e (3, 4, 5)

OUTPUT



somme 6, 9 e 12: la media è (6 + 9 + 12) ÷ 3 = 9

Problema 10 — La media fra i −1

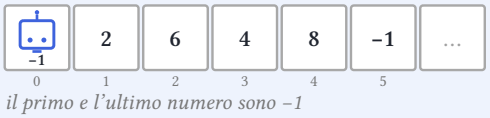
★★★

Scenario. L'ultima lista è quasi tutta di numeri positivi: solo il primo e l'ultimo numero sono −1.

Obiettivo. — Scrivi sullo scontrino la media aritmetica di tutti i numeri che si trovano fra i due −1.

ESEMPIO

INPUT



il primo e l'ultimo numero sono −1

OUTPUT



2 + 6 + 4 + 8 = 20, e 20 ÷ 4 = 5

Complimenti!

Se sei arrivato fin qui, hai già la testa da programmatore.

2 Come funziona un computer

Il robot della prima sezione esegue un vocabolario piccolissimo di istruzioni: avanza, gira, leggi, scrivi, se, ripeti, stop. Un computer fa esattamente la stessa cosa, solo che il suo vocabolario è un po' più ricco e lui esegue miliardi di istruzioni al secondo. In questa sezione apriamo la scatola e scopriamo come è fatto dentro e come riesce a rappresentare **qualsiasi** cosa — numeri, lettere, immagini — usando soltanto due simboli.

2.1 Dentro la scatola

Un computer è un insieme di pezzi che collaborano. I più importanti sono cinque.

CPU **Il processore**
è il «cervello»: esegue le istruzioni una dopo l'altra, fa i calcoli e decide cosa fare in base ai confronti. È il nostro robot, ma fatto di silicio.

RAM **La memoria di lavoro**
è la memoria veloce e temporanea: contiene i dati e i programmi mentre il computer li sta usando. Quando spegni il computer, la RAM si svuota.

DISCO **Il disco rigido**
è la memoria lenta e permanente: conserva file, programmi e foto anche quando il computer è spento. Prima di usarli, vengono copiati nella RAM.

GPU **La scheda video**
è un processore specializzato nei calcoli grafici, cioè nel disegnare immagini e animazioni. La usano anche i videogiochi e l'intelligenza artificiale.

MB **La scheda madre**
è la piastra che collega tutto: processore, RAM, disco, scheda video e periferiche comunicano tra loro attraverso i suoi circuiti.

Altre componenti, un po' meno protagoniste:

- **Alimentatore** — trasforma la corrente di casa nella corrente giusta per i componenti.
- **Ventole e dissipatori** — raffreddano processore e scheda video, che scaldano molto.
- **Porte e connettori** — USB, HDMI, prese audio: il punto di contatto con il mondo esterno.
- **Scheda di rete** — collega il computer a Internet e agli altri computer.
- **Periferiche** — tastiera, mouse, monitor, stampante: servono a dare comandi e a vedere i risultati.

Il flusso è sempre lo stesso: le periferiche mandano dati al processore, che li elabora usando la RAM come appoggio, li conserva sul disco e rimanda i risultati fuori. È lo stesso schema del robot: `leggi()` per ricevere, la memoria per lavorare, `scrivi()` per comunicare.

2.2 Il bit: l'alfabeto del computer

Acceso o spento

Dentro un computer non c'è spazio per sfumature: ogni minuscolo interruttore è o **acceso** o **spento**, cioè vale **1** oppure **0**. Questa unità minima di informazione si chiama **bit**.

Un bit da solo può dire solo due cose, quindi non basta. Mettendo insieme **otto** bit otteniamo un **byte**: con otto caselle da 0/1 possiamo scrivere $2^8 = 256$ combinazioni diverse. Ecco perché il byte è l'unità di

misura più usata per la memoria: un byte per un carattere, milioni di byte (megabyte, gigabyte) per una foto o un film.



Un byte: ogni casella vale un bit, e sotto c'è il «peso» della sua posizione. Questa sequenza vale 64 + 1 = 65.

Mettersi d'accordo

La sequenza `01000001` è soltanto una fila di otto bit: di per sé non significa niente. Se decidiamo che è un **numero**, vale 65; se decidiamo che è una **lettera**, è la lettera `A`. Il computer non lo sa: siamo noi a stabilire come leggere quei bit.

Per questo esistono le **convenzioni condivise**, cioè formati e protocolli: regole scritte una volta e rispettate da tutti.

- Per i caratteri: **ASCII**, **Unicode**, **UTF-8**.
- Per le immagini: **PNG**, **JPEG**, **GIF**, **SVG**.
- Per i suoni: **MP3**, **WAV**.
- Per i documenti: **PDF**, **DOCX**.
- Per far parlare i computer tra loro: **TCP/IP**, **HTTP**, **Wi-Fi**.

Senza un accordo, gli stessi bit sarebbero incomprensibili. È come lo scontrino del robot: `45` significa «somma», ma solo perché noi e il robot avevamo concordato di leggere lì i numeri.

Come si scrivono i numeri

Per contare usiamo dieci cifre, da 0 a 9, e il valore di una cifra dipende dalla sua posizione: nel numero 237, il 2 vale 200, il 3 vale 30 e il 7 vale 7. Il computer fa lo stesso, ma con **due** cifre: si chiama sistema **binario**.

Ogni posizione vale una potenza di 2: la più a destra 1, poi 2, 4, 8, 16, 32 e così via. Dunque `1011` non è «milleundici»: è

$$1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 11.$$



Il numero binario 1011 letto con i suoi pesi: equivale a 11.

Anche qui servono standard condivisi: gli **interi** si codificano in complemento a due (un trucco per rappresentare i numeri negativi usando il primo bit come segno), mentre i numeri con la virgola seguono lo standard **IEEE 754**, che li scompone in segno, esponente e mantissa. Proprio per come è fatto questo standard, nei programmi capita che `0.1 + 0.2` non faccia esattamente `0.3`: è una conseguenza del modo in cui i numeri vengono memorizzati, non un errore.

Come si scrivono i caratteri

Le lettere sono numeri, come tutto il resto. La **tabella ASCII** è la convenzione più famosa: assegna un numero da 0 a 127 a lettere, cifre, segni di punteggiatura e comandi. Ecco qualche esempio.

Decimale	Binario	Carattere
32	`00100000`	spazio
48	`00110000`	`0`
57	`00111001`	`9`
65	`01000001`	`A`
90	`01011010`	`Z`

Decimale	Binario	Carattere
97	`01100001`	`a`
122	`01111010`	`z`

Due curiosità utili: la `A` vale 65 e la `a` vale 97, cioè differiscono di 32 — esattamente il peso di un singolo bit. Maiuscole e minuscole sono parenti strettissime. E le cifre `0` … `9` non valgono 0…9: la cifra `0` vale 48, così si distingue dal numero 0. Quando nel robot leggevamo un numero e non una lettera, era perché noi avevamo deciso di interpretare quei bit come numeri.

L'ASCII copre solo 128 simboli e non basta per accenti, lettere greche o emoji. Per questo è nato **Unicode**: una tabella enorme che assegna un numero a ogni carattere di ogni lingua. UTF-8 è il formato che trasforma questi numeri in byte. Per esempio `è` è il numero 232, e in UTF-8 occupa due byte; una faccina come `😀` ne occupa quattro.

Come si parla al processore

Rimane la domanda più importante: come faccio a dire al processore di eseguire l'azione x?

Il processore possiede un **instruction set** (insieme di istruzioni): un vocabolario ristretto di operazioni come **carica**, **somma**, **confronta**, **salta**. È esattamente il vocabolario del robot, un po' più ampio. Ogni istruzione è a sua volta un numero, con una parte che dice **cosa fare** e parti che dicono **su quali dati**.

Scrivere quei numeri a mano si chiama **codice macchina**. Per non impazzire, si usano nomi leggibili — `MOV`, `ADD`, `JMP` — che corrispondono uno a uno alle istruzioni: è il linguaggio **assembly**. Un programma chiamato **assembler** traduce i nomi in numeri. I linguaggi ad alto livello come **Python** sono ancora più comodi: li scriviamo in italiano/inglese, e un **interprete** o un **compilatore** li traduce, passo dopo passo, nelle istruzioni del processore.

Storicamente, prima delle tastiere, i programmi si preparavano su **schede perforate**: cartoncini con dei fori. Foro = 1, niente foro = 0. Le schede venivano lette da una macchina e diventavano istruzioni. Anche lì, tutto era ridotto a due soli simboli.

La memoria come una griglia

Immagina la memoria come una lunghissima striscia di caselle, identica a quella del robot. Ogni casella contiene un byte e ha un **indirizzo**: la prima è la 0, poi la 1, la 2, e così via.



C'è però una differenza importante rispetto al robot: il robot può solo fare un passo alla volta, mentre il processore può chiedere **direttamente** il contenuto di un indirizzo qualsiasi. Per questo si dice **accesso casuale** (random access): la memoria risponde subito, non importa se la casella è la 3 o la 30 000.

Qui nasce il concetto di **variabile**: invece di ricordarci l'indirizzo 1 048 576, diamo a quella casella un nome, per esempio `x`, e ci scriviamo dentro un valore.



Tre variabili: tre nomi che puntano a tre caselle della memoria. Da qui in avanti useremo proprio questa idea per capire le variabili di Python.

3 Introduzione a Python

Abbiamo visto che un computer esegue istruzioni e conserva dati in caselle di memoria con un indirizzo. Python è il linguaggio con cui gli scriveremo queste istruzioni: molto più leggibile del codice macchina, ma comunque rigoroso. Prima di iniziare, quattro regole di base — ovvie per chi programma, fondamentali per chi comincia.

> **Obiettivo.** — Il computer esegue **esattamente** quello che è scritto, non quello che intendevi. Python non perdona le distrazioni: sii preciso.

- **I caratteri contano uno a uno.** Maiuscole e minuscole sono diverse: `print` e `Print` sono due cose distinte, e `x` non è `X`. Una lettera sbagliata e il programma si ferma.
- **Gli spazi a inizio riga contano.** Il rientro (indentazione) dice a Python quali istruzioni appartengono a un blocco. Tutte le righe dello stesso blocco devono avere lo **stesso** rientro; usa sempre lo stesso numero di spazi (la convenzione è 4). Se sbagli, Python dà errore.
- **I simboli vanno chiusi e abbinati.** Ogni parentesi aperta va chiusa: `(` con `)`, `[` con `]`, `"` con `"`. Le righe `if`, `else` e `while` finiscono con i due punti `:`, che introducono il blocco rientrato.
- **Una istruzione per riga.** Python legge una riga alla volta; per scrivere una nota che il computer deve ignorare si usa `#`.

Un programma si scrive in un file (per esempio `main.py`) e si esegue dall'alto verso il basso, una riga dopo l'altra, proprio come le istruzioni del robot.

3.1 Variabili

Una **variabile** è un nome attaccato a una casella della memoria. È la stessa idea di `a`, `b` e `c` nella pancia del robot, con due novità: possiamo scegliere il nome che vogliamo e ogni casella contiene un valore che possiamo leggere e cambiare.

Per scrivere un valore dentro una variabile si usa il segno `=`, che si legge «diventa» e **non** è l'uguale della matematica:

```
x = 5
```

Da leggere: «nella casella x metti il valore 5».



Poi possiamo usare quel nome per rileggere il valore e per calcolare:

```
x = 5
y = 3
z = x + y
print(z)
```

`z` *diventa 8. Il valore di destra viene calcolato e poi riposto nella casella di sinistra.*

Le operazioni disponibili sono le solite: `+` (somma), `-` (sottrazione), `*` (moltiplicazione), `/` (divisione) e `%` (resto della divisione, per esempio `7 % 2` fa `1`).

I nomi delle variabili seguono poche regole: possono contenere lettere, cifre e il trattino basso `_`, non possono iniziare con una cifra e non possono contenere spazi. Quindi `numero` e `numero_1` vanno bene, `1numero` no. Ricorda: `nome` e `Nome` sono due variabili diverse.

3.2 Liste

Spesso non basta una casella sola: serve una **sequenza** di valori. In Python si chiama **lista** e si scrive tra parentesi quadre. I valori stanno in caselle **contigue**, una attaccata all'altra, numerate a partire da **0** — proprio come le caselle del robot.

```
v = [10, 20, 30]
```



Per leggere un elemento si scrive il nome della lista e, tra parentesi quadre, la sua **posizione** (indice):

```
print(v[0])
print(v[2])
```

Stampa 10 e poi 30. Il primo elemento è `v[0]`, non `v[1]`!

Python sa contare anche da destra: `v[-1]` è l'ultimo elemento, `v[-2]` il penultimo. E `len(v)` restituisce quanti elementi contiene la lista (`3`, in questo caso). Per aggiungere e togliere elementi si usano due metodi:

```
v.append(40)
v.remove(20)
```

`append(40)` aggiunge 40 in fondo; `remove(20)` elimina la prima casella che contiene 20. La lista diventa `[10, 30, 40]`.

3.3 Decidere: if ed else

Come il robot con **se** e **altrimenti**, Python prende decisioni con `if` e `else`. La differenza è che la condizione va scritta con `==` per chiedere «sono uguali?», mentre `=` resta l'assegnazione.

```
x = 7
if x > 5:
    print("x è grande")
else:
    print("x è piccolo")
```

Il blocco dentro `if` e quello dentro `else` sono rientrati: è il rientro a dire dove comincia e dove finisce ciascun blocco. La riga dell'`if` e quella dell'`else` terminano con `:`. I confronti disponibili sono:

- `==` uguale a
- `!=` diverso da
- `<=` minore o uguale
- `>=` maggiore o uguale

- `<` minore di
- `>` maggiore di
- `and` e `or` per unire più condizioni
- `not` per negare una condizione

3.4 Ripetere: while e break

Il robot aveva `ripeti:` per ripetere all'infinito e `stop` per fermarsi. In Python c'è `while`, che ripete **finché** una condizione è vera, e `break` per uscire dal ciclo.

```
i = 0
while i < 3:
    print(i)
    i = i + 1
```

Stampa 0, 1, 2 e poi si ferma: quando `i` diventa 3 la condizione è falsa.

La grande differenza rispetto al robot è che `while` vuole una condizione sulla riga. Se la condizione non diventa mai falsa, il ciclo non finisce più: per questo dentro il ciclo bisogna **cambiare** qualcosa (nel nostro esempio, `i` cresce). Quando serve uscire prima, si usa `break`:

```
i = 0
while True:
    if i == 5:
        break
    i = i + 1
```

`while True` ripete per sempre, esattamente come `ripeti:`; `break` è lo `stop` del robot.

3.5 Parlare con l'utente: print

Il robot comunicava scrivendo sullo scontrino con `scrivi(x)`. In Python lo scontrino è lo schermo e l'istruzione si chiama `print`:

```
print("Ciao!")
print(3 + 4)
x = 10
print(x)
```

`print` accetta più valori separati da virgola e li stampa uno dopo l'altro:

```
x = 3
y = 5
print("La somma è", x + y)
```

Stampa: `La somma è 8`.

Il testo tra virgolette si chiama **stringa** e va sempre racchiuso tra `"` oppure `'`. I numeri, al contrario, si scrivono senza virgolette: `print(5)` stampa il numero, `print("5")` stampa il testo.

3.6 Dal robot a Python: cosa cambia

La logica è la stessa; cambia il modo di scriverla. Questa tabella riassume le differenze principali.

Robot	Python	Attenzione
`a = a / b`	`a = a / b`	Qui `=` assegna: la casella a sinistra riceve il valore di destra.
`se a = b:`	`if a == b:`	Nel robot `=` confronta; in Python confronta `==`, mentre `=` assegna.
`altrimenti:`	`else:`	Le parole chiave cambiano, l'idea no.
`ripeti:`	`while True:`	Il ciclo infinito del robot diventa un while sempre vero.
`stop`	`break`	Esce dal ciclo più vicino.
`scrivi(x)`	`print(x)`	Lo scontrino del robot diventa lo schermo.
`leggi(x)`	`x`	Nel robot si copia dalla casella; in Python si legge direttamente il nome.
memoria a, b, c	qualsiasi nome	I nomi li scegliamo noi, non più solo lettere singole.
rientro di 2 spazi	rientro (di solito 4)	In Python il rientro è obbligatorio e fa parte della sintassi.

3.7 Esercizi introduttivi

Piccoli esercizi per prendere confidenza con la sintassi. Scrivi il codice in un file, eseguilo e confronta il risultato con la soluzione.

Esercizio 1 — Il primo saluto



Scrivi un programma che stampi sullo schermo la scritta `Ciao, mondo!`.

SOLUZIONE

```python
print("Ciao, mondo!")
```

Esercizio 2 — Una variabile e la sua stampa

★☆☆

Crea una variabile `x` con dentro il numero `7`, poi stampala.

SOLUZIONE

```
x = 7
print(x)
```

Esercizio 3 — La somma

★☆☆

Crea due variabili `a` e `b` con due numeri a piacere e stampa la loro somma.

SOLUZIONE

```
a = 3
b = 5
print(a + b)
```

Esercizio 4 — Primo e ultimo

★☆☆

Crea la lista `v = [10, 20, 30]` e stampa il primo e l'ultimo elemento, usando le parentesi quadre.

SOLUZIONE

```
v = [10, 20, 30]
print(v[0])
print(v[2])
```

Esercizio 5 — Scambio di valori

★★☆

Le variabili `a` e `b` contengono due numeri. Scambia tra loro i valori e stampali alla fine. (Suggerimento: serve una terza variabile di appoggio.)

SOLUZIONE

```
a = 1
b = 2
c = a
a = b
b = c
print(a)
print(b)
```

Esercizio 6 — Trova l'errore

★★☆

Questo programma non funziona. Trova l'errore e correggilo.

```
x = 5
if x = 5:
    print("uguale")
```

SOLUZIONE

Nella condizione dell' `if` serve il confronto `==`, non l'assegnazione `=`:

```
x = 5
if x == 5:
    print("uguale")
```

3.8 Esercizi più avanzati

Ora che la sintassi è chiara, ragioniamo con cicli e liste — gli stessi problemi che risolvevi con il robot, ma scritti in Python.

Esercizio 7 — Conta da a a b

★★☆

Leggi due numeri nelle variabili `a` e `b` e stampa tutti i numeri interi da `a` a `b`, estremi inclusi.

SOLUZIONE

```
a = 3
b = 8
i = a
while i <= b:
    print(i)
    i = i + 1
```

Esercizio 8 — Il massimo

★★☆

Data una lista di numeri, trova il valore più grande e stampalo.

SOLUZIONE

```
v = [4, 9, 1, 7]
massimo = v[0]
i = 1
while i < len(v):
    if v[i] > massimo:
        massimo = v[i]
    i = i + 1
print(massimo)
```

Esercizio 9 — È crescente?

★★☆

Stabilisci se una lista di numeri è crescente (ogni elemento è maggiore o uguale al precedente) e stampa `True` oppure `False`.

SOLUZIONE

```
v = [1, 3, 3, 8]
crescente = True
i = 1
while i < len(v):
    if v[i] < v[i - 1]:
        crescente = False
    i = i + 1
print(crescente)
```

Esercizio 10 — Al contrario

★★☆

Data una lista, costruisci una nuova lista con gli stessi elementi in ordine inverso, poi stampala.

SOLUZIONE

```
v = [1, 2, 3, 4]
inv = []
i = len(v) - 1
while i >= 0:
    inv.append(v[i])
    i = i - 1
print(inv)
```

Esercizio 11 — Somma e media

★★☆

Data una lista di numeri, calcola e stampa la loro somma e la loro media.

SOLUZIONE

```
v = [2, 4, 6]
somma = 0
i = 0
while i < len(v):
    somma = somma + v[i]
    i = i + 1
print(somma)
print(somma / len(v))
```

Esercizio 12 — Quanti pari

★★★

Conta quanti numeri pari ci sono in una lista e stampa il risultato. Usa l'operatore `%`.

SOLUZIONE

```
v = [1, 2, 3, 4, 5, 6]
pari = 0
i = 0
while i < len(v):
    if v[i] % 2 == 0:
        pari = pari + 1
    i = i + 1
print(pari)
```

Esercizio 13 — Il primo negativo

★★★

Scorri una lista finché non trovi il primo numero negativo, poi stampalo. Usa `break` per uscire dal ciclo.

SOLUZIONE

```
v = [5, 3, -2, 8]
i = 0
while i < len(v):
    if v[i] < 0:
        break
    i = i + 1
print(v[i])
```



Fine del percorso

Dal vocabolario del robot alle prime righe di Python: la strada è aperta.