Fondamenti di informatica

Michela Paolucci (ricevimento su prenotazione)
http://www.disit.org/drupal/?q=node/7020

LEZIONE 03/03/20

ALGORITMO:
Descrive un procedimento per risolvere una classe di problemi in un numero finito di passi descritti in modo rigoroso.
Quindi è un **elenco finito e ordinato di istruzioni.**
Formalizzare un algoritmo significa trascrivere l'algoritmo da una scrittura informale ad una scrittura formale o matematica.
Può essere eseguito dall'uomo, dalle macchine o da entrambi, e deve essere **comprensibile** dall'esecutore.

DATI (input) → ALGORITMO → RISULTATI (output)

Le proposizioni che costituiscono l'algoritmo sono formate da: **operazioni che devono essere eseguite, oggetti sui quali devono essere effettuate le operazioni**

ELABORATORE:
Apparecchiatura **AUTOMATICA**, **DIGITALE** ed **ELETTRONICA** in grado di **eseguire gli algoritmi** sulla base di dati diversi. Capace quindi di trasformare dati in ingresso.
Richiede un linguaggio di programmazione (es. Python) generalizzato e compatibile con l'elaboratore, costituito da strutture linguistiche definite.
Quindi istruzioni fornite in modo da esso eseguibile.
L'elaboratore è **flessibile.**

DATI → ELABORATORE (ALGORITMO) → RISULTATI

INFORMATICA:
"Scienza che studia l'elaborazione delle informazioni e le sue applicazioni; più precisamente l'i. si occupa della rappresentazione, dell'organizzazione e del trattamento automatico della informazione. Il termine i. deriva dal francese "Informatique" (composto di INFORMATion e automatIQUE, «informazione automatica») e fu coniato da P. Dreyfus nel 1962. "
Enciclopedia Treccani

(VEDI SCHEMA SLIDE 62)

PROGRAMMA:
Un insieme di istruzioni ordinate secondo uno schema, che corrisponde all'implementazione di un algoritmo, quindi la **traduzione di un algoritmo in un linguaggio comprensibile dall'elaboratore** (calcolatore)

Un programma è registrato nella memoria di un elaboratore.

La SINTASSI di ogni riga di codice deve essere assolutamente corretta.

RAPPRESENTAZIONE

L'obbiettivo è dare ad un algoritmo una forma tale che possa essere eseguita da un elaboratore, attraverso un linguaggio di programmazione.

Dobbiamo formalizzare in maniera logica i concetti attraverso:

Strumenti:
Algebra di Boole: **algebra binaria**, solo due possibili variabili (es. 0/1, on/off, vero/falso...) (non legata esclusivamente all'informatica)
Flow Chart: (**diagrammi di flusso**) disegni/schemi che rappresentano graficamente il ragionamento alla base dell'algoritmo, rendono più chiara ed immediata la comprensione del problema

NOTE:

LEZIONE 09/03/20

LINGUAGGIO PYTHON

Linguaggio di alto livello, ovvero che necessita di un'elaborazione umana prima di essere eseguiti. Più comprensibili dall'uomo e meno dalla macchina, al contrario di quelli di basso livello. **Versatili** per vari tipi di macchina senza necessità di modifiche, o con piccole modifiche.

I linguaggi di alto livello devono essere trasformati in linguaggi di basso livello per essere eseguiti dalle macchine, tramite due passaggi: **INTERPRETAZIONE** e **COMPILAZIONE**

1) **interprete** legge il programma e lo esegue, trasformando le istruzioni in azioni, alternando i due passaggi (istruzione-azione)

2) **compilatore**: legge il programma di alto livello e lo traduce in basso livello
Chiameremo il programma di alto livello "**CODICE SORGENTE**" ed il programma tradotto "**CODICE ESEGUIBILE**"

Il compito del programmatore è quindi solo quello di creare il codice sorgente e verificare la corretta escuzione del programma.

Python è considerato un linguaggio interpretato perché i programmi Python sono eseguiti da un interprete

L'interprete si una a **LINEA DI COMANDO** o in modo **SCRIPT**

CONSOLE = linea di comando
con questo metodo bisogna scrivere una linea per volta nella console e premere Invio, così da far interpretare una linea per volta

FILE → script
si può scrivere il programma in un file e farne eseguire il contenuto dall'interprete, indicandogli dove trovare questo file e come è nominato
Il file che può essere letto da Python ha estensione ".py"
(VEDI SLIDE 78)

Risulta più conveniente scrivere i comandi con la linea di comando per creare un programma semplice e testarlo, e poi salvare il programma in uno script così da riutilizzarlo o modificarlo facilmente, perché appena viene chiuso l'ambiente Python il programma viene perso.
Per programmi più complessi non è sufficiente utilizzare la linea di comando.

ISTRUZIONI COMUNI A TUTTI I LINGUAGGI DI PROGRAMMAZIONE:

- **input**
- **output**
- **matematiche:** operazioni semplici
- **condizionali:** controllo di alcune condizioni ed esecuzione della sequenza di istruzioni adeguata
- **ripetizione:** ripetizione di un'azione, magari con qualche variazione

LINGUAGGI FORMALI E NATURALI

I linguaggi naturali sono le lingue comunemente utilizzate.

I linguaggi formali sono linguaggi progettati per specifiche applicazioni (es. matematica, chimica...), sono molto rigidi nella sintassi.

Le **regole sintattiche** si dividono in due categorie: **TOKEN** e **STRUTTURA**

Token = elementi base del linguaggio (es. 137+9, H_20)
Struttura della dichiarazione = modo in cui i token sono disposti

Grazie ad una corretta sintassi posso capire la semantica della frase.

I **linguaggi formali** **NON POSSONO AVERE AMBIGUITÀ**, ciascuna dichiarazione ha un determinato significato indipendentemente dal contesto
Risultano inoltre più **concisi**, e assolutamente **letterali**

1. TIPI DI DATI

Ciascun dato è proprio di un **tipo**.
Un TIPO è rappresentato dai valori che può rappresentare e dalle operazioni che possono essere effettuate su tali valori.

Tipo di dato primitivo: tipo di dato che vien emesso a disposizione dal linguaggio

vediamone alcuni:

- **int** (numeri interi, anche con segno)
- **float** (numeri a virgola mobile o **foating point**)
- **str** (stringhe: sequenze di caratteri, tipo "hello world")

esistono tipi di dati già esistenti come quelli sopracitati, ma posso anche creare io nuovi tipi
Se non sono sicuro del tipo di un valore usato è possibile chiederlo all'interprete

es.
>>> type (2.4) (richiesta)
< class “float” > (risposta)
>>> type (“18”) (richiesta (se ci sono “” sarà sempre str))
< class “str” > (risposta)
(VEDI SLIDE 92)

Altri tipi di dati:

- **bool** (Boolean)

introdotti con l’algebra di Boole, in Python vengono espressi tramite “True” o “False”
es.
>>> print (10>9) (richiesta)
True (risposta)
>>> print (10<9)
False
(VEDI SLIDE 94)

2. COSTANTI E VARIABILI

COSTANTE: valore che non varia nel corso della computazione, quindi un elemento al quale viene associato sempre lo stesso valore, che non deve poter cambiare nemmeno per errore.
es. se voglio calcolare una circonferenza utilizzando le prime 10 cifre del π come costante questa non dovrà mai cambiare.

VARIABILE: valore che varia nel corso della computazione

In Python a livello sintattico non c’è distinzione tra costante e variabile, è il programma a dover sapere cosa sta utilizzando.
Normalmente **per indicare le costanti si utilizzano le lettere Maiuscole**.

(IMP!!! VEDI SLIDE 96-97-98)

3. OPERATORI ARITMETICI

Sono simboli speciali che rappresentano i calcoli fondamentali come addizione, moltiplicazione ecc. (+ , - , * , /)
** = elevamento a potenza

= è l’operatore di assegnazione
VEDI ESEMPIO SLIDE
LEZIONE 12/03/20

ASSEGNAZIONE

Sintassi → nomeDiVariabile ,= (segno di uguaglianza), valore

Nel momento in cui **assegno** un valore ad una variabile occupo un "posto" nella memoria del computer, che posso immaginare come se riservassi una delle limitate celle che costituiscono questa memoria.

es. a=4
(ho occupato la cella a con il valore 4, se cambio la variabile ho sostituito il contenuto della cella, non aggiunto)

Pyhton darà errore se ad esempio scrivessi area=base*ALTEZZA senza aver definito in precedenza la variabile "base" e la, in questo caso, costante "ALTEZZA"

REGOLE DEI NOMI DI VARIABILE

- sequenza di caratteri alfabetici, numerici
- non si possono usare simboli o spazio, è ammesso solo "_" (undersore)
- il primo carattere non può essere un numero
- Pyhton è un CASE SENSITIVE, quindi scrivere "area" o "Area" sono due variabili diverse
- è consigliabile non utilizzare gli accenti nei nomi delle variabili
- esistono delle **"parole riservate"** che non possono essere usate come nomi di variabili perché per Python definiscono delle regole di linguaggio

>>> import keyword
>>> keyword.kwlist
["False", "None", "True", "and", "as", "assert""async", "await", "break", "class", "continue", "def", "del", "elif", "else", "except", "finally", "for", "from", "global", "if", "import", "in", "is", "lambda", "nonlocal", "not", "or", "pass", "raise", "return", "try", "while", "with", "yield"]

CONCETTO DI *ISTRUZIONE*

"In informatica, elemento della programmazione con cui si richiede al computer, attraverso un codice prestabilito, l'esecuzione di una determinata operazione (leggere i dati in ingresso, effettuare un calcolo, una selezione, ecc.)
Enciclopedia Treccani
quindi una stringa di codice che l'interprete possa eseguire
ESPRESSIONE: combinazione di variabili, valori e operatori. O anche singole variabili (con valore precedentemente indicato) o singoli valori sono considerati un'espressione

ISTRUZIONE DI ASSEGNAZIONE: creare una variabile assegnandole un valore
ISTRUZIONE STAMPA: print
ISTRUZIONE TYPE: richiesta di indicare il tipo di variabile

MODALITÀ INTERATTIVA E MODALITÀ SCRIPT

Scrivendo nelle righe di codice ho la possibilità di inserire e provare pezzi di codice uno alla volta prima di inserirli in uno script.

In modalità interattiva posso vedere i risultati immediati di ogni linea di codice, inserendoli in uno script questi codici non hanno alcun effetto immediato. In uno script le espressioni non hanno effetti visibili.

Se non vi è un comando che renda visibile il risultato (es. print) nello script, quando viene inserito in python ancora non verrà visualizzato nulla, ma il programma avrà memorizzato le istruzioni. **(VEDI SLIDE 116-117)**

Uno script quindi contiene normalmente una sequenza di istruzioni.

ORDINE DELLE OPERAZIONI

Python usa lo **stesso ordine di precedenza della matematica.**

Acronimo **PEMDAS** (Parentesi, Elevamento a potenza, Moltiplicazione e Divisione, Addizione e Sottrazione)
Gli operatori con la stessa priorità vengono valutati da sinistra verso destra (eccetto la potenza).

Non è possibile effettuare operazioni matematiche sulle stringhe, anche se il loro contenuto è un numero.

L'operatore + funziona sulle stringhe, attaccando il contenuto di queste, in questo caso il + si comporta da **operatore di concatenamento**

es.
>>>primo= "bagno"
>>>secondo= "sciuma"
>>>print (primo+secondo)
< bagnoschiuma >
Anche l'operatore * funziona con le stringhe, ma solo se una stringa viene moltiplicata per un numero intero, non quindi per un'altra stringa. Il * funziona quindi da **ripetitore** della stringa.

es.
>>>print (“Ciao”*3)
< CiaoCiaoCiao >

si comporta quindi analogamente alla sua funzione matematica

COMMENTI

Man mano che il programma cresce di dimensioni è buona cosa inserire dei **commenti (note) che spieghino** in linguaggio naturale il significato di determinate righe di codice o che riassumano il contenuto del programma in un determinato punto. Conviene quindi spiegare ogni tanto il contenuto e il motivo di righe di codice particolarmente complesse.

Queste note vengono **indicate dal simbolo # che le precede.**
Quindi qualsiasi cosa scritta dopo il simbolo # viene ignorata da Python.

Fondamentali se si lavora in team ad un programma.

DEBUG

Si dice debuging **la ricerca e la correzione dei bug** (errori).
In programmazione ci possono essere tre tipi di bug:

- errori di sintassi
- errori in esecuzione
- errori di semantica

Ci sono degli strumenti per effettuare il debug, tra questi l’esecutore stesso aiuta ad individuare alcuni errori.

1. ERRORI DI SINTASSI

Python individua gli errori di sintassi mostrando un messaggio di errore ed interrompendo quindi l’esecuzione, rendendo impossibile proseguire.

Il messaggio d’errore indica in quale riga è stato commesso l’errore così da poter andare facilmente a correggerla. A volte indica addirittura un consiglio per una possibile correzione.
(VEDI ESEMPIO SLIDE 124)

2. ERRORI IN ESECUZIONE (o “runtime bug”)

Questo tipo di errore non appare finché il programma non è in esecuzione.

Sono detti anche eccezioni perché indicano che è accaduto qualcosa di eccezionale durante la programmazione (es. si è cercato di dividere per 0)

Li tratteremo quando creeremo programmi complessi.

3. ERRORI DI SEMANTICA

Python non indica gli errori di semantica ed esegue ugualmente il programma, perché Python non sa cosa il programmatore vuole ottenere, esegue e basta.
Il risultato non sarà quindi quello desiderato.

Può accadere se il significato del programma (semantica) è scorretto. Per individuare quindi questi errori occorre **cercare i difetti del programma già in esecuzione**.

Per evitare questi errori è IMPORTANTE PRESTARE ATTENZIONE DURANTE LA FASE DI ANALISI.

Per alcuni programmare e il debuging sono la stessa cosa. In quanto nel programmare è necessario evitare e risolvere gli errori, che sempre si verificano.

OPERATORI ED ESPRESSIONI

Un'espressione è una combinazione di costanti e riferimenti a variabili tramite operatori.

Gli operatori sono classificati per **tipo di operazione:**

ARITMETICI	• Divisione / • Modulo (resto) % • Somma + • Differenza - • Prodotto * • Elevamento a potenza ** • Op. composto Incremento += • Op.c. Decremento -= • Op.c. *= • Op.c. /= • Parte intera / /
	• Minore <

RELAZIONALI	• Maggiore > • Maggiore uguale >= • Minore uguale <= • Uguaglianza == • Diverso != • Identità is • negazione di identità is not
LOGICI	• Congiunzione && (and) • Negazione ! (not) • Disgiunzione || (or)

OPERATORI ARITMETICI

```
>>> var=10
>>> var += 2
>>> print(var)
12
```

ATTENZIONE AGLI OPERATORI COMPOSTI!!!
L'uguale da solo (=) è l'operatore di assegnazione!

OPERATORI RELAZIONALE

<valore 1> <operatore relazionale> <valore 2>

Sono operatori **binari**
Il risultato fornito da Python è di tipo binario:

- True se la testata (affermazione,espressione) è corretta
- False se non è corretta

OPERATORI LOGICI

Introdotti dall'algebra di Boole
Si introducono quindi le tabelle di verità

not:
operatore **unario** (opera su una sola variabile)

A	not A
False	True
True	False

VEDI SLIDE 141

and:
operatore **binario**
A and B è una funzione logica che mi restituisce True solo se entrambe le espressioni sono True

A	B	A and B
False	False	False
False	True	False
True	False	False
True	True	**True**

Ho quindi due colonne d'ingresso

IMP. VEDI ESEMPI SLIDE 142!!!

NOTE:

LEZIONE 16/03/20

or:
operatore binario
A or B è una funzione logica che mi restituisce True se almeno uno delle due espressioni (o operandi) è True, quindi False se entrambe le espressioni sono False.

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

Anche qui ho due colonne in ingresso

IMP. VEDI ESEMPI SLIDE 144!!!

STAMPARE: USO DI PRINT

Il comando print(…) indica all'esecutore di stampare ciò che viene inserito nelle parentesi.
es.
>>> print("ciao")
ciao
>>> print(la temperatura oggi è: \n10° centigradi
la temperatura oggi è:
10° centigradi

usando quindi il simbolo "\n" nel testo, indico di andare a capo (ATT. NON /)
\n è detto **operatore di formato**, che permette di **formattare la stampa in uscita.**

OPERATORE DI FORMATO (format specifier) PER STRINGHE

VEDI SLIDE 148-149

%
Se a sinistra dell'operatore % vi è una stringa, allora % diventa un operatore di formato.
Sintassi → stringaConteneteInfromazioniDiFormato %

Le informazioni di formato contenute nella stringa solo gli operatori di formato: %d e %.3f

```
>>> quantita = 748
>>> peso = 235.746432
>>> print("Numero di oggetti: %d Peso totale: %.3f" % (quantita, peso))
Numero di oggetti: 748 Peso totale: 235.746
```

essendoci due indicazioni di formato indico poi i due valori a cui dovranno fare riferimento (quantità, peso).

%d = gestione degli interi
quindi quando vorrò stampare un intero scriverò %d, ed indicherò a quale intero mi riferisco alla fine dopo il % tra parentesi, dove appunto indico i valori
(nel nostro esempio l'intero indicato è la variabile quantita)

%.Nf = gestione dei float
quindi per stampare un float indicherò con %.Nf l'operazione e a N sostituirò il numero di cifre che mi interessa prendere dopo la virgola, ed ancora andrò ad indicare la variabile alla quale mi riferisco nelle parentesi dopo il %
(nel nostro esempio chiedo di considerare 3 cifre dopo la virgola (%.3f), riferite alla variabile peso (245.746))

Conversione	Significato
d	Numero intero decimale con segno.
i	Numero intero decimale con segno.
o	Ottale senza segno.
u	Decimale senza segno.
x	Esadecimale senza segno (minuscolo).
X	Esadecimale senza segno (maiuscolo).
e	Numero in virgola mobile, in formato esponenziale (minuscolo).
E	Numero in virgola mobile, in formato esponenziale (maiuscolo).
f	Decimale in virgola mobile.
F	Decimale in virgola mobile.
g	Lo stesso di "e" se l'esponente è più grande di -4 o minore della precisione, "f" altrimenti.
G	Lo stesso di "E" se l'esponente è più grande di -4 o minore della precisione, "F" altrimenti.
c	Carattere singolo (accetta interi o stringhe di singoli caratteri).
r	Stringa (converte ogni oggetto Python usando `repr()`).
s	Stringa (converte ogni oggetto Python usando `str()`).
%	Nessun argomento viene convertito, riporta un carattere "%" nel risultato.

se scrivo &10.3f indico con il dieci il fatto che voglio utilizzare 10 caratteri, ed il tre il numero di cifre decimali. Il punto occupa una casella.
Se il valore non occupa tutti gli spazi richiesti (in questo caso 10) mi verranno mostrati gli spazi vuoti.

INDICATORE	ESEMPIO DI VISUALIZZAZIONE	DESCRIZIONE
%d	2 4	Numeri interi
%5d	_ _ _ 2 4	Aggiungi spazi a sx (5 ampiezza totale)
%05d	0 0 0 2 4	Aggiungi zeri a sx (5 ampiezza totale)
%f	1 . 2 4 8 1 6	Numeri float
%.2f	1 . 2 5	Due cifre dopo il punto decimale, arrotondando
%7.2f	_ _ _ 1 . 2 5	Aggiungi spazi a sx (7 ampiezza totale), arrotondando
%s	H e l l o	Stringhe
%9s	_ _ _ _ H e l l o	Ampiezza totale 9, stringa allineata a dx
%-9s	H e l l o _ _ _ _	Ampiezza totale 9, stringa allineata a sx
%d%%	2 4 %	Per visualizzare il segno di % si utilizza %%
%+6d	_ _ _ _ + 2 4	Usando il segno +, i numeri positivi si visualizzano con il +

ho utilizzato _ per indicare gli spazi vuoti, pyhton mostra solo spazi vuoti

(VEDI ESEMPI SLIDE 152-153)

STAMPARE STRINGHE E NUMERI

Per stampare più stringhe insieme useremo come visto in precedenta l'operatore +, che fungerà da operatore di concatenazione (anche se il contenuto delle stringhe è numerico), mentre lo useremo tra due interi per avere la somma di questi due.

Come posso stampare insieme stringhe e interi o float?
Python mi restituirà errore se provo semplicemente a utilizzare il + tra una stringa e un intero o un float.

Dovrò attuare una operazione di **conversione di tipo (CAST)**
→ Ad esempio dicendo a Pyhton di trattare le variabili intere o float come stringhe, attraverso la funzione str(), precisamente "+str(...)+"

es.
>>>print("Oggi a Firenze ci sono " + str(20) + "°C"
Oggi a Firenze ci sono 20°C
(se non inserisco io gli spazi Pyhton non li mette!)
es.2
>>>gradi_oggi=20
>>>print("Oggi a Firenze ci sono " + str(gradi_oggi) + "°C"
Oggi a Firenze ci sono 20°C
(VEDI SLIDE 156)

→ In determinati casi è possibile anche trasformare stringhe in interi o float, attraverso le funzioni int() e float().
es.
>>>id = int("1734")
>>>price = float("17.34")

Può essere utile se un ipotetico utente inserisse un numero in una stringa, che poi nel programma conviene usare come intero o float.
(VEDI SLIDE 158-159)

STAMPA A CONSOLE

Ricorrere a **format**
Sintassi → stringa.format(dato in ingresso1, dato in ingresso2, dato in ingresso n...)

Il numero di dati in ingresso corrisponde al numero di variabili inserite nella stringa

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> 'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N', longitude='
-115.81W')
'Coordinates: 37.24N, -115.81W'
```

il format concatena anche tipi diversi, senza ricorrere al CAST

LEZIONE 19/03/20

ISTRUZIONI

rivedi concetto di istruzione p.6-7

BLOCCO DI ISTRUZIONI

Un blocco di istruzioni è un insieme di istruzioni che complessivamente formano una macroistruzione, può essere considerato quindi l'equivalente di un'istruzione:

- Ogni istruzione può essere **scomposta in istruzioni più semplici** finché non si arriva ad una istruzione che non può essere ulteriormente scomposta (approccio **top-down**).
- Ogni istruzione può essere **raggruppata all'interno di un'istruzione più complessa** (compound), fino ad arrivare al livello dell'intero programma (approccio **bottom-up**).

USO DEI FLOW CHART

Per schematizzare flussi di ragionamento, quindi sono uno strumento per visualizzare/formalizzare gli algoritmi.
I passi finiti degli algoritmi sono tradotti in *istruzioni* per il programma.

Le istruzioni servono per "dirigere" il flusso dell'esecuzione di un programma nel giusto ordine.

Per rappresentare i flow chart si fa uso di:

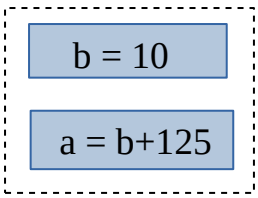
- **rettangoli** dentro ai quali vengono inserite delle espressioni e istruzioni
- **frecce e linee continue** per collegare le varie componenti
- **linee tratteggiate** per effettuare raggruppamenti (compound o blocchi di istruzioni)
- **rombi** per esprimere condizioni (OPERATORI RELAZIONALI, che mi aprono quindi due possibili strade: True o False

Per creare un compound dovrò usare l'indentazione (vedremo in seguito). **Un numero uguale di TAB (spazio) prima delle due espressioni faranno si che queste vengano eseguite in blocco.**

es.

```
if(var==6):
    b = 10
    a = b+125
```

ovvero



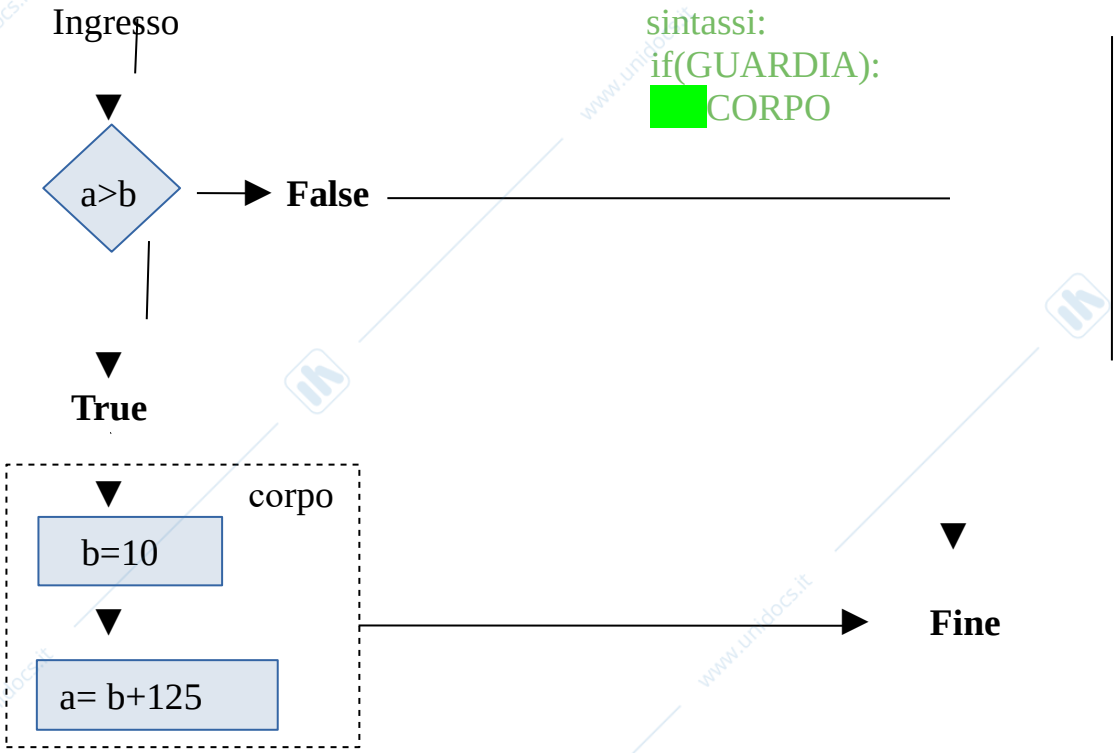
(mancano le frecce)

Cosi sto dicendo all'interprete che se var fosse uguale a 6 dovrebbe eseguire le due istruzioni scritte sotto contemporaneamente, se var fosse diverso da 6 non ne verrebbe eseguita nessuna. Quindi tratto i blocchi di istruzioni in maniera unitaria. Quindi i TAB (o spazi) condizionano la sintassi vincolando l'ordine di associazione dei termini.

ISTRUZIONE CONDIZIONALE: if

Le istruzioni condizionali permettono di decidere la direzione da prendere nel flusso di esecuzione, in base al valore in ingresso.

If condiziona l'esecuzione di un'istruzione detta **CORPO**, al risultato restituito da un'espressione detta **GUARDIA** (a>b nel nostro esempio).



Quindi se la guardia è True vengono eseguiti tutti i comandi del corpo, altrimenti il corpo viene ignorato.
(VEDI ESEMPIO SLIDE 178-180)

INDENTAZIONE

Una delle regole base per scrivere correttamente un programma leggibile è l'indentazione (che fa parte della sintassi) affiancata all'uso dei commenti (per la leggibilità). L'indentazione in Python determina il flusso di esecuzione delle istruzioni.

È in pratica è **l'inserimento di spazi o tabulazioni** (che normalmente vengono ignorati dall'esecutore) per stabilire eventuali gerarchie dei cicli o delle funzioni.

ESEMPI SLIDE 183

COMMENTI PT.2

- su una riga di codice (o inline): #commento
- su più righe di codice:
  ”””
  questo è un
  commento su più righe
  ”””

Quindi per scrivere commenti su più righe di codice devo mettere **tre virgolette prima e dopo il commento.**

ISTRUZIONE CONDIZIONALE: if else

L’istruzione if può essere estesa attraverso l’uso di: if else.
Così da avere due corpi alternativi che vengono valutati o meno in base al valore della guardia.

Si indica quindi un corpo di istruzioni da eseguire nel caso la guardia sia False.

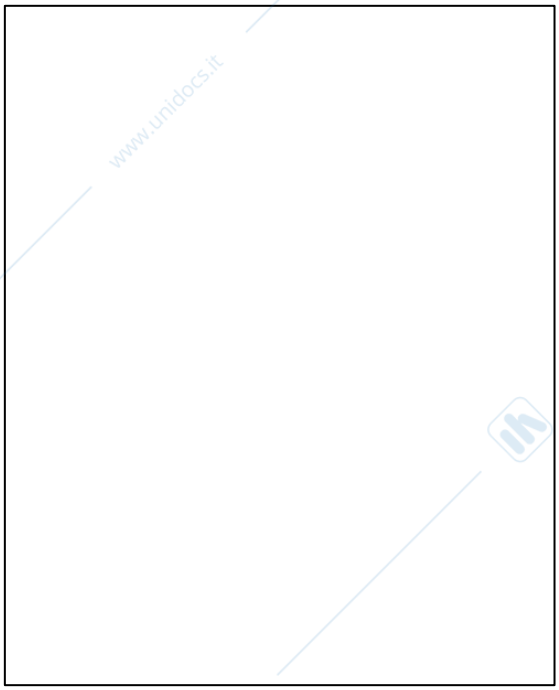
Sintassi →

```
if(GUARDIA):
    CORPO1
else:
    CORPO2
```

(disegnare qui il diagramma di flusso)

Richiamando l’esempio precedente:

```
a = 5
b = 22
if(a>b):
    a = b+125
else:
    b = 10
    a = b+125
    print(ciao)
ciao
```



CICLI WHILE E FOR

ISTRUZIONI DI ITERAZIONE

Le istruzioni di iterazione permettono di eseguire ripetitivamente un corpo di istruzioni finché non si verifica una certa condizione sui valori delle variabili del programma.

for:

sintassi→ **for** nomedivariabile **in range (**condizione di inizio**,**guardia**)**
corpo

	Quando viene valutata	Descrizione
condizione_inizio ciclo (es. count=0)	Prima di eseguire il ciclo (una sola volta)	Usata per l'**inizializzazione** dell'indice del ciclo
Guardia (es. count=9)	Viene controllata PRIMA dell'esecuzione di OGNI iterazione	Usata per verificare l'**uscita** dal ciclo for
Incremento (es. count = count+1) *sottinteso*	Incremento effettuato DOPO OGNI iterazione	Usato per **incrementare** l'indice del ciclo

es.

```
>>>for count in range (0,10)
...         print(count)
0
1
2
ecc..
9
>>>
```

(il range sarà la guardia che imporrà all'esecutore di smettere di ripetere l'operazione print(count), in questo caso **quando count sarà <10, non uguale**)

Quindi il for fa si che il l'istruzione scritta nella riga successiva a for (in questo caso print(count)) si ripeta per tutto il range con un incremento (nel nostro caso di 1).

L'istruzione che impone l'incremento di count è implicita nell'iterazione for.

Se scrivessi solo for count in range (10), quindi senza mettere la condizione iniziale, pyhton darebbe per scontato che il range sia da 0 a <10. Si dice che per Default la condizione iniziale è 0.

VEDI ESEMPI SLIDE da 193 a 199 *(disegna flow chart e prova su py)*

while:

anche while è una modalità di iterazione, quindi un comando che crea un ciclo, è quindi esattamente corrispondente a for come risultato ottenibile, cambia solo la sintassi.

Sintassi →

condizione iniziale
while nomedivariabile guardia
    corpo
    incremento

(almeno una è necessaria, quella riferita alla variabile che userò nel while)

es.

```
count=0
sum=0
while count < 10
    sum = sum + count
    count = count + 1
```

a differenza del for con while è obbligatorio indicare il valore dell'incremento, che in questo caso non è scontato e che se non indicato rimane 0

VEDI ESEMPIO SLIDE 201-202-203 *(disegna flow chart)*

ESEMPIO DELL'UTILIZZO DI for: calcolo del fattoriale (prova su py)

INTERAZIONE DA CONSOLE

Per far interagire ipotetici utenti diversi dal programmatore con il programma attraverso la console.

Usiamo la funzione **input()** per permettere all'utente di immettere dati da tastiera, che diventeranno valori per il programma.

Input accetta un singolo argomento opzionale: una stringa che viene mostrata a video prima di leggere il valore digitato. Una volta che l'utente ha digitato un valore e premuto il tasto Invio, input restituisce il valore come stringa, assegnandola alla variabile indicata dal programmatore (es. nome).

es.

```
>>>nome= input("Inserisci il tuo nome: ")
Inserisci il tuo nome: Michela
```

VEDI ESEMPI DI INTERAZIONE

>>>print(“Ciao “+nome+”!”)
Ciao Michela!
>>>

SLIDE DA 208 A 213