Cose da sapere per l'esame di Fondamenti di informatica V1.0

Basato su su un'analisi personale dei vecchi compiti e su opinioni personali (di un informatico vero, NON ingegniere)

Matte Moni

June 12, 2023

Indice degli argomenti

0 Lista schietta di tutto

1 Comandi da sapere per forza

Di seguito sono riportati i comandi che sono presenti nella stragrande maggioranza degli esercizi proposti e senza i quali è praticamente impossibile passare l'esame:

1.1 Operazioni

Tema 1.1. Operazioni di base

Date due variabili `a,b = 0,0` come eseguire le operazioni di base fra loro (addizione, sottrazione, moltiplicazione, divisione)

Soluzione.

1. Addizione: `c = a+b`
2. Sottrazione: `c = a-b`
3. Moltiplicazione: `c = a*b`
4. Divisione: `c = a/b`

Note.

1. **La spaziatura** nelle operazioni è abbastanza arbitraria `c = a+b` e `c = a + b` sono entrambi accettabili
2. **La divisione per 0** porta ad un errore da parte dell'interprete a runtime, ed è uno degli **edge cases** più comuni che vanno sempre testati quando c'è una divisione.
3. Se si vuole assegnare il valore dell'operazione ad una delle variabili che ci concorrono, si può usare **la forma ridotta (e molto più stilosa e leggibile)** del tipo: `a += b` che equivale a svolgere `a = a + b`

Tema 1.2. Operatori binari e unari

Date due variabili `a,b = True, True` come utilizzare gli operatori binari e unari (and, or, not)

Soluzione.

1. And: `c = a and b`
2. Or: `c = a or b`
3. Not: `c = !b` oppure `c = not b`

Nota. Di solito questi operatori si usano considerando come "variabili" delle valutazioni di verità, es:

```
x,y = 2,1

if x > y and y == 1
    do something
```

equivale a svolgere:

```
x,y = 2,1
a = x > y
b = y == 1

if a and b:
    do something
```

(Per quanto questa seconda forma sia inusuale, il codice funziona effettivamente, provare per credere)

Tema 1.3. Altri operatori utili

Date due variabili `a,b = 0,0` come eseguire altri tipi di operazioni fra loro.

Soluzione.

1. Elevazione a potenza (a^b): `c = a**b`

1.2 Liste, set e dizionari

Quello che viene espresso in questa sezione riguarda la liste, con una breve spiegazione di cosa sono i set e i dizionari. **Nella sezione delle solutions to common problems**, ci sono le applicazioni principali di set e dizionari (una per ognuno) che al prof. piacciono tanto e che in effetti sono molto carine e si usano anche nella vita reale.

Tema 1.4. Creare una lista

Come creare una lista di elementi

Soluzione. `A = [ ]` crea una lista vuota, `A = [x,y,z] crea una lista con gli elementi x,y,z`

Nota. Se A **non** è una lista, non è possibile utilizzare su A i metodi propri delle liste, es:

```
A = 0
print(A[0])
```

restituirà errore

Tema 1.5. Creare (e utilizzare) set e dizionari

Come creare altre strutture dati utili (e utilizzarle) (Set e dizionari)

Soluzione.

1. **Set (insieme)**:

   ```
   A = {0, 1, 2}
   ```

   crea un set, ovvero un insieme di elementi unici, elementi uguali vengon rimossi e l'ordine non è definito. Non si può accedere ai set con la sintassi `A[i]`.
   Non ho visto mai necessario un largo utilizzo dei set nei compiti vecchi, in ogni caso alcune informazioni utili sono:
   - Per aggiungere/rimuovere elementi si usano i metodi `A.add(elemento)` e `A.remove(elemento)`
   - Si possono eseguire operazioni fra insiemi come: differenza: `C = A-B`, unione: `C = A.union(B)`, intersezione: `C = A.intersection(B)`
2. **Dizionario**:

   ```
   A = {0:"a", 1:"b", 2:"c"}
   ```

   crea un dizionario, ovvero un associazione chiave, valore, che permette di cercare un valore al suo interno in base alla chiave. Le chiavi devono essere uniche, mentre i valori no.
   Non ho visto mai necessario un largo utilizzo dei set nei compiti vecchi, in ogni caso alcune informazioni utili sono:
   - Aggiungere/rimuovere elementi: `A[chiave] = valore` e `A.pop(chiave)`
   - Restituire tutte le chiavi: `A.keys()`
   - Restituire tutti i valori: `A.values()`
   - Restituire coppie chiave, valore: `A.items()`
   - Restituire un valore data la chiave: `A.get(chiave)` oppure `A[chiave]`

Nota. Non tutti i comandi presenti nei temi successivi si applicano a set e dizionari, alcuni sì. In generale consiglio di aprire l'IDE (il programma che si usa per scrivere codice) e vedere quali comandi si applicano a degli oggetti di tipo set o dizionario.

Tema 1.6. Lunghezza di una lista

Data una lista di elementi `A = [x,y,z]` come conoscere la sua lunghezza.

Soluzione. `len(A)` ritorna la lunghezza di A.

Note.

1. len è una funzione built-in di Python e come tale si applica senza usare la sintassi oggetto.funzione(parametro), ma usando solo funzione(parametro).
2. Dato che len ritorna la lunghezza di una lista, in generale restirutirà un valore che equivale alla posizione **successiva** a quella dell'ultimo elemento della lista, dato che una lista di lunghezza n ha elementi che vanno dalla posizione 0 alla posizione n-1.

Tema 1.7. Accedere alle liste

Data una lista di elementi `A = [x,y,z]` come accedere ad un elemento al suo interno.

Soluzione.

1. Metodo base: `A[i]` accede all'elemento i-esimo.
2. Accesso dal fondo: `A[-i]` accede all'elemento i-esimo **a partire dal fondo**.
3. Accesso ad un sottoinsieme di elementi della lista: `A[i : j]` accede agli elementi dall'i-esimo al j-esimo. `A[i :]` e `A[: j]` accedono agli elementi dall'i-esimo all'ultimo e dal primo al j-esimo rispettivamente.

Nota. La prima posizione di una lista è la posizione 0, mentre **l'ultima posizione** è la posizione lunghezza-1, quindi il primo elemento sarà `A[0]`, mentre l'ultimo sarà `A[len(A)-1]`. (Provare ad accedere ad `A[len(A)]` darà un errore di out of bounds)

Tema 1.8. Aggiungere e rimuovere elementi

Data una lista di elementi `A = [x,y,z]` come aggiungere e rimuovere elementi

Soluzione.

1. Aggiungere un elemento "elem" in testa: `A.append(elem)`
2. Aggiungere un elemento "elem" ad una posizione "pos": `A.insert(pos, elem)`
3. Rimuovere un elemento ad una posizione "pos": `A.pop(pos)`
4. Rimuovere il primo elemento di valore noto "val" (se esiste): `A.remove(val)`

Nota. Il metodo pop ritorna l'elemento rimosso, mentre il metodo remove no, per cui un'istruzione del tipo `a = A.pop(pos)` assegnerà ad a il valore contenuto in `A[pos]`, mentre `a = A.remove(elem)` assegnera ad a il valore "None" (e non ha quindi senso di essere usato così).

Tema 1.9. Ordinare gli elementi in una struttura dati

Data una struttura dati A (che imponga un ordine negli elementi, quindi ad esempio NON un set), come ordinare gli elementi al suo interno

Soluzione.

1. Restituire una lista ordinata data una struttura dati: `B = sorted(A)`
2. Ordinare le liste (e SOLO le liste): `A.sort()`

Note.

1. Il metodo `sorted` si applica ad un qualunque elenco di valori ma **ritorna sempre una lista**, quindi è possibile ad esempio usare il metodo per ordinare gli elementi in un set e ottenere una lista con gli stessi elementi all'interno del set ma ordinati.
2. Il metodo `sorted` può ricevere i parametri `reversed` e `key` che permettono di decidere in base a cosa ordinare la lista e se ordinarla all'inverso. Non ho mai visto essenziale il metodo `key`, `reversed` invece è abbastanza intuitivo e può tornare utile: `A.sort(reversed =true)`

1.3 Matrici

Nella maggior parte degli ultimi compiti c'è sempre un esercizio su una matrice. Solo in pochi c'è effettivamente da creare una matrice m*n, (vedi 2). Nella maggior parte dei casi la matrice è ricevuta come input e va solo scorsa correttamente.

Tema 1.10. Accedere ad una matrice

Data una matrice di elementi `A = [[1,2,3],[3,2,1]]`, come accedere ad un elemento al suo interno.

Soluzione. Si usa esattamente la stessa sintassi che per i vettori monodimensionali ma ricordandosi di usare un doppio puntatore, quindi per accedere all'elemento a_{ij} si dovrà accedere ad `A[i][j]`. Valgono tutti gli accorgimenti espressi per i vettori monodimensionali.

Esempio. Creare una funzione (vedi 1.6), che data una matrice, stampa uno ad uno tutti gli elementi della stessa:

```
def stampa_matrice(A):
    for i in range(len(A)): #Ciclo che scorre le colonne
        for j in range(len(A[0])):  #Ciclo che scorre la riga della colonna in questione
            print(A[i][j])
```

Tema 1.11. Aggiungere e rimuovere elementi

Data una matrice di elementi `A = [[1,2,3],[3,2,1]]`, come aggiungere e rimuovere elementi

Soluzione. Il metodo più semplice per capire come fare ad aggiungere e rimuovere elementi è quello di vedere una matrice come una lista di liste. Una volta compreso questo si possono usare tutti i metodi che si usano per le liste classiche con le dovute accortezze.
Non si aggiungono singoli elementi ma intere righe (o colonne), e per farlo di solito si considera la nostra matrice A e si esegue il comando: `A.append(B)` dove B è un vettore della lunghezza consona.
Allo stesso modo **Non** si rimuovono singoli elementi ma intere righe (o colonne), e per farlo basta eseguire il comando di `A.pop(pos)`, per rimuovere l'intera riga alla posizione "pos".

1.4 Condizioni

Tema 1.12. if, elif, else

Come eseguire parti di codice in base a certe condizioni usando if, elif ed else.

Soluzione. La sintassi delle scelte condizionali in Python è:

```
if(condizione):
    do something
elif(condizione):
    do something
else:
    do something
```

Con alcuni accorgimenti:

1. elif è **opzionale** e ce ne può essere anche più di 1.
2. else è **opzionale** ma ce ne può essere al massimo 1.

Nota. È possibile **annidare** (ovvero mettere una dentro l'altra) le condizioni. Per non fare confusione il consiglio è di suddividere il problema in sottoproblemi partendo da quello della condizione più **interna**. Una volta che quella parte funziona si comincia a risalire con le condizioni verso l'esterno.

1.5 Cicli

In generale tutti gli esercizi sono svolgibili ciclando solo con cicli while **ma** con i cicli for spesso il codice risulta più snello e la soluzione è più semplice da pensare. In più usando il ciclo for **non** si hanno problemi dovuti all'errato utilizzo del contatore. Il prof usa quasi sempre cicli for.
Come già espresso per le condizioni, anche i cicli si possono **annidare**, e vale esattamente lo stesso consiglio, ovvero, cercare di capire prima cosa deve fare il ciclo più interno per poi spostarsi a quelli più esterni una volta che quello funziona.

Tema 1.13. Ciclo for su una struttura dati

Come usare il ciclo for per ciclare su una struttura dati

Soluzione. Al contrario di quello che succede in altri linguaggi, in Python il ciclo for si usa spessissimo nella forma **standard** che permette di ciclare su una struttura dati::

```
A = [1,2,3]
for elem in A:
    do something
```

Svolgendo il contenuto del ciclo for per ogni elemento presente nella struttura dati (una lista in questo caso).

Note.

1. **Non si possono eliminare elementi** da una struttura dati su cui si sta ciclando in questo modo. (In caso usare una nuova struttura dati dove mettere gli elementi interessati)
2. A seconda del tipo di `elem`, le operazioni che si possono svolgere sullo stesso chiaramente cambiano, è importante sapere il tipo di elementi che ci sono nella lista prima di eseguire questo tipo di ciclo.

Tema 1.14. Ciclo for con range

Come usare il ciclo for su un range di valori deciso con la funzione range().

Soluzione. La funzione range presenta 3 versioni:

1. versione con 1 parametro: `range(max)` che restituisce i numeri da 0 a max-1 con distanza di 1 fra un numero e l'altro.
2. versione con 2 parametri: `range(inizio, max)` che restituisce i numeri da inizio a max-1 con distanza di 1 fra un numero e l'altro.
3. versione con 3 parametri: `range(inizio, max, step)` che restituisce i numeri a partire da inizio a max-1 con distanza di step fra un numero e l'altro.

Il ciclo for con l'utilizzo del range permette di ciclare su un contatore che viene automaticamente inizializzato all'interno del ciclo ed incrementato della giusta quantità ad ogni ciclo. La sintassi è la seguente:

```
for i in range(inizio, max, step):
    do something
```

Note.

1. In generale la versione con 3 parametri non è mai sbagliata, ma è meglio evitarla se possibile, ad esempio è meglio evitare una sintassi del tipo `for i in range(0, len(A), 1)`, che equivale a `for i in range(len(A))`
2. In questo caso stiamo ciclando con un contatore, quindi sebbene l'idea sia spesso quella di usare il ciclo per scorrere una struttura dati, va notato che il nostro parametro formale adesso è effettivamente un contatore (es.i) che è un numero intero, quindi se per esempio vogliamo eseguire una funzione su ogni elemento di una lista `A` dovremo riferirci ai suoi elementi usando la forma `A[i].funzione()`.
3. Questa sintassi ci permette anche di ciclare anche a ritroso, usando una forma del tipo:

```
for i in range(max-1, inizio-1, -step):
    do something
```

Tema 1.15. Ciclo while

Come usare il ciclo while

Soluzione. In Python l'unico ciclo oltre al for è il ciclo while, il ciclo while è un ciclo precondizionale, ovvero un ciclo in cui la condizione viene verificata prima di eseguire il ciclo stesso. Questo vuol dire che può essere comodo per le situazioni in cui è possibile che il ciclo venga eseguito anche 0 volte. La sintassi è la seguente:

```
while(condizione verificata):
    do something
```

Note.

1. Una **rule of thumb** per il ciclo while è che si cicla sempre per vero, è buona norma riscrivere la condizione se si sta ciclando per falso.
2. Questo tipo di ciclo è comodo anche per scrivere cicli "infiniti", che terminano solo grazie ad un intervento esterno o comunque a certe circostanze (anche grazie al comando `break` (spiegato in seguito)), ad es:

```
while(True):
    do something
    if(condizione):
        break
```

3. A differenza del ciclo for, qui la gestione del contatore è completamente lasciata al programmatore, di conseguenza una delle sintassi più comuni che si vedono è la seguente:

```
i,n = 0,10
while (i < n)
    do something
    i += 1
i = 0 #Solo se la i puo' servire di nuovo in futuro
```

Tema 1.16. Break

Come usare il comando break per uscire prematuramente da un ciclo

Soluzione. NOTA BENE: Non l'ho visto usato neanche in un compito, neanche quando secondo me ci stava bene, sarebbe da informarsi se dice di non usarlo, se fosse così, vedere la nota della sintassi del break "fatto in casa" con il flag.
Il comando break si usa per uscire da un ciclo prematuramente, senza dover passare per forza dal controllo in cima al ciclo. **Si può usare sia in cicli for che in cicli while**. La sintassi è:

```
while(condizione verificata):
    do something
    if(condizione):
        break
```

Nota. l break è molto comodo, anche se in generale può essere sostituito nella maggior parte dei casi dall'utilizzo apposito di una variabile flag, come segue:

```
flag = False
while(condizione verificata and flag == False):
    do something
    if(condizione):
        flag = True
```

(Nota personale, a me il break piace abbastanza di più e non vedo perchè demonizzarlo, però per passare l'esame me ne fregherei della mia opinione)

1.6 Funzioni

In generale è buona norma usare una funzione ogni volta che si inserisce una funzionalità nel programma.
Le funzioni permettono di evitare la ripetizione di codice, rendono il codice più leggibile e facile da debuggare.
Suddividere il problema in sottoproblemi più semplici risolvibili uno dopo l'altro permette di concentrarsi su una cosa alla volta in modo da sapere subito cosa funziona già e cosa no, e sapere quindi dove apportare modifiche al programma se presenta problemi (l'ultima cosa aggiunta a rigor di logica).

Tema 1.17. Creare una funzione

Come creare una funzione in python in modo corretto, rispettando le convenzioni e con tutte le buone norme del caso.

Soluzione. La sintassi per dichiarare una funzione di base è:

```
def function_name(parameter):
    do something
    return something
```

I parametri passati in input possono essere da 0 fino a potenzialmente infiniti.
Il return è opzionale, una funzione che non **ritorna** niente si dice una funzione **void**, mentre una funzione che ritorna qualcosa si dice una funzione di tipo equivalente al tipo del valore che ritorna.

Note.

1. In python se il nome di una funzione è composto da più parole si usa la **snakecase** notation, ovvero si separano le parole con un _ e si scrive l'intero nome **in minuscolo** (es. `nome_funzione`).
2. La funzione può essere scritta in modo che i parametri abbiano dei valori di default, che permettono di richiamare la funzione anche con un numero minore di parametri, usando la sintassi:

   ```
   def function_name(parameter1, parameter2 = default_value):
       do something
       return something
   ```

   Chiaramente vanno messi prima i parametri senza valore di default, se presenti.
   Non l'ho mai visto necessario negli esercizi, ma può essere una mossa di stile e può aprire a soluzioni interessanti in specifici casi.

Tema 1.18. Richiamare una funzione

Come richiamare una funzione presente nel codice.

Soluzione. La sintassi per richiamare una funzione è:

```
function_name(parameter)
```

Il numero di parametri deve essere coerenti con quelli che sono definiti dalla funzione. Colui che chiama la funzione deve rispettare **il contratto** che chi l'ha scritta ha deciso, sia per quanto riguarda il numero, che per quanto riguarda l'ordine, dei parametri che devono essere passati.

Nota. Esiste un modo per specificare anche i parametri in ordine diverso rispetto a quello "di base", ed è eseguito con la sintassi:

```
function_name(nome_parametro1 = valore, nome_parametro2 = valore)
```

Tuttavia sconsiglio di usarlo in generale se è evitabile (e negli esercizi non l'ho mai visto necessario). Può risultare utile se si usano tanti parametri con valori di default e se ne vuole specificare qualcuno anche in disordine.

1.7 Classi e oggetti

In Python ogni cosa è un oggetto (che è quello che dice il prof.), tuttavia di base gli oggetti semplici (interi, numeri in virgola mobile, booleani ecc.) si comportano in modo abbastanza diverso da tutti gli altri oggetti.
Anche le classi built-in in python si comportano in modo diverso da quelle che l'utente può creare o che può trovare nelle librerie. Un esempio è la classe "str", built-in, con cui si creano le stringhe, che segue chiaramente sintassi diversa da quella che verrà qui esposta.
Le classi sono di solito utilizzate per rappresentare qualcosa che esiste anche nel mondo reale.
L'idea alla base della classe è quella di avere una parte di codice che permetta di creare degli **oggetti** di un certo tipo che hanno certe caratteristiche (**attributi**) e sono in grado di svolgere delle azioni (**metodi**).
Ogni oggetto di una classe è unico e differente da ogni altro, però hanno tutti in comune (in generale) le caratteristiche che li caratterizzano e le azioni che possono compiere, e per questo il tipo di un oggetto definisce in ogni caso per cosa quello si usa, a discapito di come sarà poi inizializzato a runtime.

Esempio. Una classe di nome **Gatto** permette di creare oggetti di tipo Gatto, ognuno con caratteristiche: **nome, età, colore**, e azioni che può compiere: **miagola e graffia**. Ogni gatto può avere nome diverso o miagolare in modo diverso, ma se io ho un oggetto di tipo Gatto so per certo che presenterà un nome e che sarà in grado di miagolare.

Tema 1.19. Come creare una classe

Come creare una classe in python in modo corretto, rispettando le convenzioni e con tutte le buone norme del caso.

Soluzione. La sintassi per creare una classe di base è:

```
class ClassName:

    def __init__(self, parameter):
        self.parameter = parameter

    def method_name(self, parameter):
        do something
```

Per le classi si usa sempre la prima lettera maiuscola e la **camelcase** notation ovvero tutte le parole che compongono il nome della classe inizieranno in maiuscolo e non ci sarà nessuno spazio fra una parola è l'altra.
`__init__` è detto **costruttore** e serve per istanziare un oggetto della classe con eventuali parametri passati in input quando si crea l'oggetto (NON si richiama mai direttamente il metodo `__init__`, lo si chiama solo implicitamente quando si crea l'oggetto (vedi 1.7)).
I parametri di `__init__` e di tutti gli altri eventuali metodi della classe, iniziano sempre con il parametro `self` dopodichè possono esserci da 0 a infiniti altri parametri.
Tutti i metodi della classe, se presenti, seguono le regole delle funzioni classiche con la differenza che il primo parametro deve essere sempre self, ma questo parametro NON sarà mai passato direttamente quando verrà richiamato il metodo

Tema 1.20. Importare classi e moduli all'interno di altri file

Come strutturare il codice usando più file .py in modo da usare le funzioni e le classi messe a disposizione dagli stessi.

Soluzione. Esistono varie sintassi per importare solo specifiche parti di moduli che sarebbero da preferire se siamo al corrente di come li useremo, tuttavia il metodo generico per moduli e file (che contengono funzioni e/o classi) in generale è:

```
import file_name
import module_name
```

Così facendo per richiamare direttamente le funzioni provenienti dal modulo o dal file sarà sufficiente usare la sintassi `file_name.method_name()`.

Nota. Per comodità si può decidere di assegnare un nome alternativo con la seguente sintassi:

```
import file_name as better_file_name
import module_name as better_module_name
```

Tema 1.21. Istanziare ed usare gli oggetti

Come si istanzia un oggetto di una classe precedentemente creata e come si utilizzano correttamente i suoi metodi e i suoi parametri.

Soluzione. Gli oggetti si istanziano (in parole povere creano) ed usano come segue:

```
oggetto1 = ClassName("parametro")  #Crea un oggetto di tipo class_name, con parametro "parametro"
oggetto1.method_name() #Richiama il metodo method_name con parametro self
```

```
var1 = oggetto1.parameter_name #Assegna a var1 il valore nel parametro parameter_name
```

Note. Alcuni accorgimenti da notare quando si usano gli oggetti sono:

1. il costruttore si richiama richiamando il nome della classe (e non richiamando direttamente il metodo `__init__()`)
2. Come si può notare nonostante tutti i metodi dell'oggetto abbiano come parametro in input `self` (vedi 1.7, in pratica questo NON va mai passato direttamente, ma è automaticamente passato dall'interprete quando si inizializza l'oggetto e quando si richiamano i suoi metodi.
3. Nell'esempio il metodo `method_name()` è un metodo void, quindi viene richiamato senza assegnare il risultato, tuttavia possono esistere metodi che ritornano dei valori.
4. 2 oggetti inizializzati esattamente con gli stessi parametri NON rappresentano lo stesso oggetto.
5. Se un metodo ritorna un altro oggetto con campi o metodi si può usare una sintassi del tipo `oggetto.method_name().other_method_name()` a cascata.
6. Spesso ho notato che il prof fa creare le classi ma NON richiede di creare effettivamente oggetti delle stesse all'interno di eventuali altre classi e/o funzioni. Tuttavia è utile sapere come funziona per testare il codice e per capire come strutturare la classe in modo corretto.

2 Solutions to common problems

Tema 2.1. Rimuovere duplicati da una lista

Data una lista, come rimuovere tutti i valori duplicati al suo interno

Soluzione. ci sono vari metodi con cui si potrebbe risolvere questo problema, tuttavia il più interessante è di sicuro quello di sfruttare il concetto di set, che non può contenere ripetizioni. Usando la funzione built-in `set()` è possibile trasformare una lista in un set, il che eliminerà automaticamente i duplicati. A quel punto sarà sufficiente usare la funzione built-in `list()` per trasformare il set di nuovo in una lista.

Esempio. Creare una funzione che, data in input una lista A, ritorna una lista uguale ma rimuovendo i duplicati:

```
def remove_duplicates(A):
    return list(set(A))
```

Tema 2.2. Creare una matrice $m * n$

Come creare una matrice di dimensione m*n con n ed m noti, m numero di righe ed n numero di colonne.

Soluzione. In python al contrario di altri linguaggi non è possibile dichiarare una matrice di dimensione indefinita fin da subito. Per questo ci sono varie logiche che si potrebbero usare per risolvere questo problema. Il metodo preferito dal Prof. è quello di inizializzare un vettore di dimensione indefinita per poi andare a comporre uno alla volta gli m vettori di dimensione n da inserire al suo interno.

Esempio. Creare una matrice di dimensione m*n random (fra 5 e 10) e riempirla di valori random (fra 10 e 100):

```
import random

A = []  #Matrice che verra' riempita di vettori
m = random.randint(5,10)
n = random.randint(5,10)

for i in range(m):
    B = []  #Ad ogni ciclo reinizializzo il vettore B e poi lo riempio
    for j in range(n):
        B.append(random.randint(10,100)) #Inserisco n numeri, uno alla volta, in B
    A.append(B) #Aggiungo in cima ad A il vettore B precedentemente creato
```

Chiaramente questa soluzione è applicabile a tutti i casi in cui si ha modo di ottenere i valori m,n e tutti quelli interni, in anticipo. (Anche senza usare la funzione random).

Tema 2.3. Creare una nuova matrice vuota

Come creare una matrice B, di dimensione corretta partendo da un'altra matrice, A, in modo veloce (senza dover usare il metodo classico per riempire una matrice di numeri)

Soluzione. L'idea è quella di creare delle righe di lunghezza nota usando la sintassi: `row = [0] *len(A)` e poi inserire tutte queste righe una ad una dentro la nuova matrice B

Esempio. Creare una classe che ritorni una nuova matrice, B, di zeri delle stesse dimensioni di una matrice, A, passata in input:

```
def matrice_zeri(A):
    B = []  #Matrice che verra' riempita di zeri
    for riga in range(len(A)):
        row = [0]*len(A[0])     #A ogni ciclo crea una nuova riga di zeri
        B.append(row)        #Inserisco la riga nella matrice B
    return B                 #Ritorno perche' e' una funzione
```

Note.

1. In generale se sono in grado di conoscere lunghezza ed altezza della nuova matrice, posso usare queste metodo per creare una matrice di zeri, anche senza averne un'altra di partenza (rimpiazzando correttamente le righe dove vengono usati `len(A)` e `len(A[0])`.
2. **E' fondamentale** che la riga `row = [0] *len[A]` sia **dentro** il ciclo for, in questo modo ad ogni passaggio viene creata una nuova posizione di memoria e non viene usata sempre la stessa. Se si mette questa riga di codice fuori, tutte le righe hanno le posizioni in colonna connesse alla stessa posizione di memoria, e ad esempio modificare la posizione `A[0] [0]` porta a modificare tutte le posizioni `A[0] [x]`

Tema 2.4. Concatenazione di stringhe

Come concatenare il contenuto di più stringhe così da formarne una singola.

Soluzione. in Python ci sono vari metodi per concatenare stringhe fra loro, in generale però il metodo secondo me più semplice è quello di usare la forma con:

```
a = f"Stringa {variabile} stringa {variabile} stringa ecc."
```

che permette di concatenare fra loro stringhe con variabili in modo molto snello.
Un'altra versione carina è:

```
a = "Stringa " + variabile + " stringa " + variabile + " stringa ecc."
```

Molto simile a quella che si usa anche in Java.
Se ho già una stringa e voglio aggiungere un'altra parte di stringa posso usare la classica sintassi `a += " stringa"` oppure `a += variabile`

Esempio. Creare una classe che rappresenta un impiegato con nome e salario ed ha un metodo che permette di ritornare una stringa contenente i suoi dati.

```
class Employee:

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def employee_information(self):
        return f"Name: {self.name}, salary: {self.salary}"
```

Tema 2.5. Usare il modulo random

ome usare il modulo random per le applicazioni che ci sono nei compiti effettivamente

Soluzione. Per usare le funzioni del modulo random questa va innanzitutto importato `import random` e successivamente le funzioni più utili sono:

1. Generare **interi positivi** in [min,max]: `random.rand(max,min)`
2. Generare valori float in [-1,1): `random.random()`
3. Generare valori float in [min,max): `min + (max-min)*random.random()`

Esempio. Scrivere una funzione che generi interi compresi fra 0 e 255:

```
import random

def generate_int():
    return random.randint(0,255)
```

Nota. Volendo si potrebbe importare solo la specifica funzione randint per una questione di efficienza, ma non è fondamentale:

```
from random import randint

def generate_int():
    return randint(0,255)
```

Tema 2.6. Lavorare con gli alberi binari di ricerca (ABR)

Come comportarsi nel caso di esercizi sugli ABR

Soluzione. In generale non ho MAI visto la richiesta di realizzare un intero ABR, tuttavia è importante conoscerne la struttura generale in modo da poterci lavorare.
Spesso viene richiesto di realizzare una variante della classe nodo:

```
class Node:
    def __init__(self, data):
         self.right = None   #Figlio destro vuoto
         self.left = None    #Figlio sinistro vuoto
         self.data = data    #Data passato come parametro

    def __str__(self)
        return f"{self.data}"    #Stringa formato human readable
```

```
    def __repr__(self)
        return f"{self.data}"    #Stringa formato completo (Coincidono perche' data e' banale in
                                                                    questo caso)
```

Con varianti sul tipo di dato che rappresenta `data`, o con più parametri che rappresentanto `data`.
Guardare il file `tree.py` messo dal prof. perchè è completo e mostra come funzionano i metodi degli alberi che poi il prof. **dà per scontato che si conoscano** nel compito e su cui ci si basi per crearne delle versioni modificate.
Se si ha chiaro come è strutturato un ABR non importa impararli a memoria. 

3 Common mistakes

Di seguito sono elencati alcuni errori comuni che vengono fatti quando si scrive codice python. Aggiornabile durante lo studio come promemoria

Booleani con minuscola

I valori booleani in python vogliono la maiuscola: `True`,`False`.

Scelta delle parentesi

Può capitare di non essere sicuri di quali parentesi si applicano ad un certo contesto. La regola semplice secondo me è:

- **Tonde**: Chiamate di funzioni o metodi.
- **Quadre**: Accesso a strutture dati (liste 90% delle volte).
- **Graffe**: in python praticamente mai (al contrario di altri linguaggi), volendo per la concatenazione di stringhe.

Tipo di oggetto sbagliato

Dato che in python tutto è considerato come un oggetto, cercare di usare dei metodi che non sono applicabili su un certo tipo di dato (ad esempio `add()` su una variabile intera) restituirà un errore del tipo "il metodo nome_metodo() non è applicabile al tipo nome_tipo".
In questo caso la cosa da fare è guardare a runtime quale tipo sta assumendo la variabile in questione, e capire perchè tale metodo non è applicabile alla stessa.

Non inizializzare una variabile

In python le variabili non vanno dichiarate prima di utilizzarle. Questo in generale vuol dire che è possibile assegnare qualunque valore ad una variabile che in precedenza non era mai stata dichiarata nel codice.
Tuttavia quello che NON si può fare è cercare di utilizzare il valore contenuto in una variabile non ancora inizializzata. Se non ho inizializzato la variabile `var1`, non posso eseguire una chiamata del tipo `var2 = var1`

Errori generici

Scrivendo codice si impara a riconsocere gli errori del codice, inclusi quelli generici o quelli che sono apparentemente significativi ma che in pratica non lo sono. Fra gli errori più comuni che portano ad errori generici ci sono:

- Indentazione sbagliata
- Parentesi aperta e mai chiusa
- Stringa SENZA virgolette
- Virgolette aperte e mai chiuse

L'interprete non riesce a gestire bene questi errori perchè cerca di interpretare codice mal scritto, spesso dando errori generici, o errori in altre righe che in realtà sono corrette.