Prendiamo uno dei programmi scritti la volta scorsa, come il calcolo dell'n-esimo numero di Fibonacci.

Un'implementazione potrebbe essere la seguente:

```
n=int(input("Scrivi l'indice n: "))
a,b=0,1
for i in range(n):
    a,b=b,a+b
print("Il numero di Fibonacci F("+str(n)+") è",a+b)
```

Riscrivi il programma in modo che risultato sia corretto per valori in ingresso sensati ed eventuali risposte di errore siano gestite da te (quindi senza che compaiano scritte rosse, indipendentemente dall'input).

4.1) Scrivi un programma che richieda un numero e reciproco (`"Scrivi un numero:"`) di questo calcoli il reciproco (`"il reciproco di … è:"`) e che sappia gestire il caso di input non numerici e il caso particolare 0 (*vedi sotto* ↓)

```
Questo programma permette di calcolare il reciproco di un numero
Scrivi un numero: 4
il reciproco di 4 è 0.25
```

```
Questo programma permette di calcolare il reciproco di un numero
Scrivi un numero: W
input non ammesso
```

```
Questo programma permette di calcolare il reciproco di un numero
Scrivi un numero: 0
non si può calcolare il reciproco di 0
```

4.2) Scrivi un programma richieda un input qualsiasi e che ne riconosca la natura, come mostrato in basso:

```
Scrivi qualcosa: 11.2
11.2 è un numero!
```

```
Scrivi un numero naturale: 5P
5P non è un numero!
```

4.3) Scrivi un programma richieda l'immissione di un numero intero positivo e che preveda gli output mostrati dai seguenti esempi: (*vedi sotto* ↓)

```
Scrivi un numero intero positivo: 10
Ok!
```

```
Scrivi un numero naturale: -1
-1 è intero ma non è positivo
```

```
Scrivi un numero naturale: 1.1
1.1 è un numero positivo, ma non è un intero
```

```
Scrivi un numero naturale: -1.1
-1.1 è un numero, ma non è né positivo, né intero
```

```
Scrivi un numero naturale: X
Ma l'hai letta la richiesta?!
```

4.4) Scrivi un programma che ripeta ossessivamente "`voglio biscotti`" fino a quando non venga effettivamente inmesso "`biscotti`" e, in questo caso, dica "`grazie!`"

4.5) Considera il seguente semplice programma:

```
n=int(input("Conta fino a…: "))
for i in range (1,n+1):
    print(i,end=" ")
```

Cerca di riscrivere il codice sostituendo il ciclo for con un ciclo $while$.

4.6) Scrivi un programma che richieda in input un intero positivo (senza gestione delle eccezioni) e che riconosca se il numero è o non è un numero primo, stampando la conclusione a schermo: "`… è un numero primo`" o "`… non è un numero primo`". Cerca di ottimizzare il programma in modo che la risposta venga fornita nel minor tempo possibile (puoi testare la bontà del codice con numeri alti).

Molto difficile la seguente implementazione (continua a leggere dopo la consegna):

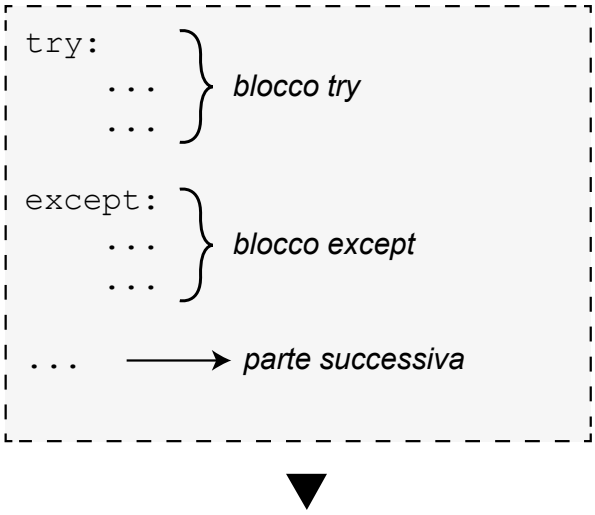
4.7A) Scrivi un programma che richieda in input un intero positivo (senza gestione delle eccezioni) e che lo scomponga in fattori primi

4.7B) Modifica il programma precedente in modo che, dopo la fattorizzazione, venga stampato a schermo il complessivo di fattori primi (eventualmente anche uguali fra loro) che compongono la scomposizione di n (ad esempio per $n = 12$ deve essere data la risposta "`nella scomposizione di 12 vi sono 3 primi`" visto che $12 = 2 \cdot 2 \cdot 3$.

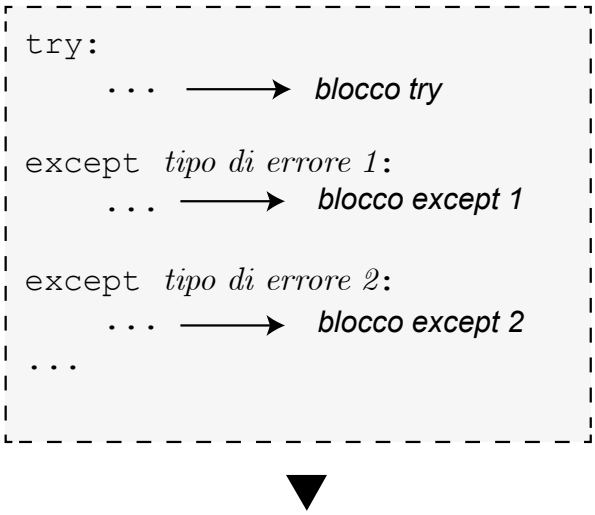
RIASSUNTO DELLE ISTRUZIONI VISTE

descrizione delle istruzioni

try except



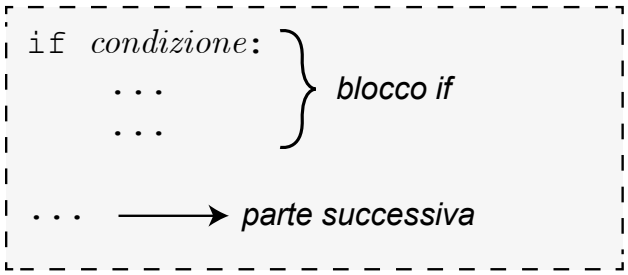
Tenta di eseguire i comandi contenuti nel blocco *try*. Se ciò non genera errori, il blocco viene eseguito integralmente altrimenti viene eseguito il codice del blocco *except*. In entrambi casi, eseguiti i blocchi, il programma passa alla parte successiva.



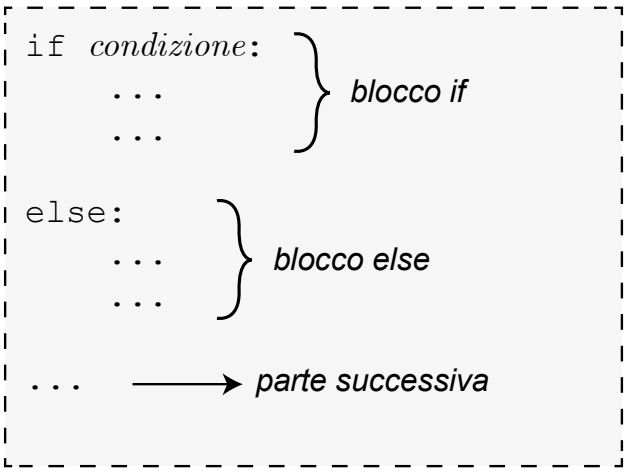
È possibile specificare eccezioni *except* diverse, a seconda del tipo di errore: i codici di errore sono molteplici, ecco tre esempi: ***ValueError*** *(la variabile usata appartiene al tipo sbagliato),* ***ZeroDivisionError*** *(si è tentato di dividere per 0),* ***IOError*** *(errore di apertura di un file)*

descrizione della clausola

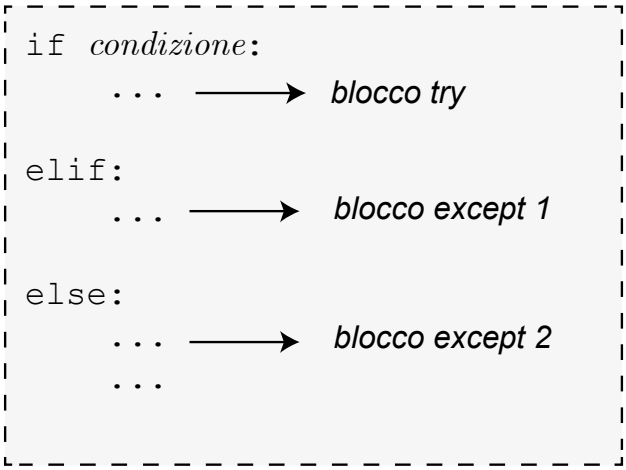
if



Se la condizione *condizione* è verificata, vengono eseguite le istruzioni indentate indicate nell'illustrazione con "blocco *if*". In ogni caso (dopo il blocco o al fallimento della condizione) il programma passa alla parte successiva del codice.



Se la condizione *condizione* è verificata, vengono eseguite le istruzioni indentate indicate nell'illustrazione con "blocco *if*", in caso contrario viene eseguito il blocco *else* (mai entrambi). In tutti i casi, eseguiti i blocchi condizionali, il programma parte alla parte succesiva.



Il blocco *else* potrebbe essere a sua volta contenere una biforcazione *if/else*. Per evitare continui rientri e un'indentatura poco chiara, si può ricorrere alla clausola *elif* (una contrazione di *else+if* che permette di mantenere il livello di rientro).

descrizione del comando

pass

```
pass
```

Non fa nulla (e passa quindi all'eventuale istruzione successiva)

Il comando **pass** viene invocato quando è previsto (o è opportuno) che il programma contenga del codice, sebbene il computer non debba svolgere alcuna azione.
Un caso del genere si presenta nella clausola *try/except,* che prevede del codice sia in caso di successo che in caso di fallimento del blocco *try*.

Esempio

```
...
try:
   print(a)
except:
   pass
```

spiegazione del codice
se nella parte di codice denotata dai puntini di sospensione è stata definita una variabile `a`*, allora viene stampato il suo contenuto, altrimenti non viene stampato nulla*

ortografia delle

condizioni

Le *condizioni* sono alla base delle clausole *if* e dei cicli *while* e consistono in enunciati logici che possono essere veri o falsi (come "a è uguale a b" oppure "a è minore di b"). Valgono i seguenti simboli:

simbolo matematico	traduzione	ortografia in Pyton
=	uguale a	`==`
≠	diverso da	`!=`
>	maggiore di	`>`
<	minore di	`<`
≥	maggiore o uguale a	`>=`
≤	minore o uguale a	`<=`
∧	e contemporaneamente (et)	`et`
∨	o anche (vel)	`and`
¬	non	`not`
V	vero	`True`
F	falso	`False`

descrizione

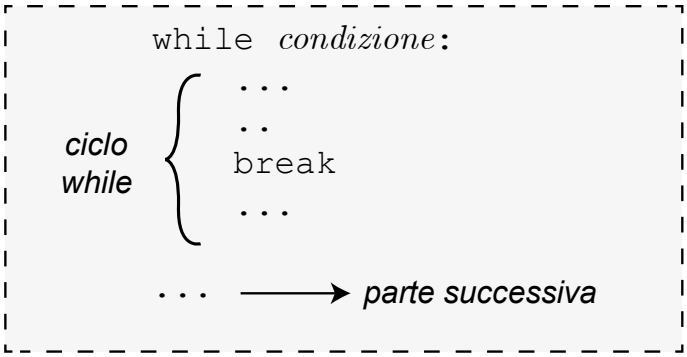
ciclo while

```
while condizione:
    ...
    ...
```
} *ciclo while*

Finché una la condizione *condizione* è soddisfatta, esegue ciclicamente il blocco di istruzioni indentate.

comandi del **ciclo while**

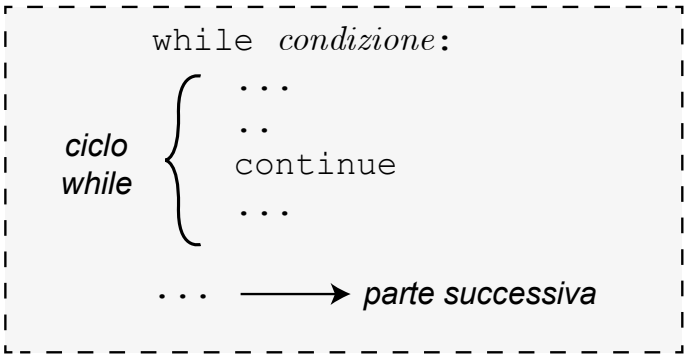
break



Se richiamato in un ciclo *while*, il comando *break* esce immediatamente Saltando direttamente alla parte del codice succesiva (esterna al ciclo).
Usualmente il comando break si trova all'interno di una clausola if.

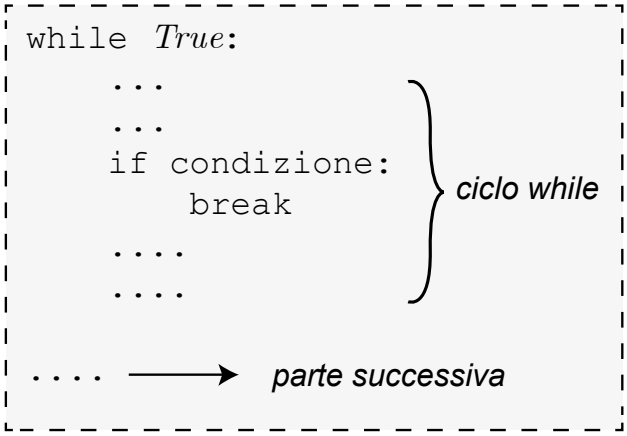
comandi del **ciclo while**

continue



Se richiamato in un ciclo *while*, il comando *continue* fa immediatamente ripartire il ciclo dall'inizio, al momento della verifica della condizione.
Usualmente il comando continue si trova all'interno di una clausola if.

specifiche del **ciclo while**



In Python *True* sta per "vero": la condizione *True* è quindi sempre soddisfatta e il ciclo *while* scritto sopra è virtualmente infinito. Cicli del genere vengono normalmente controllati da una o più clausole *if* che prevedono un comando *break* (in modo da uscire dal ciclo).