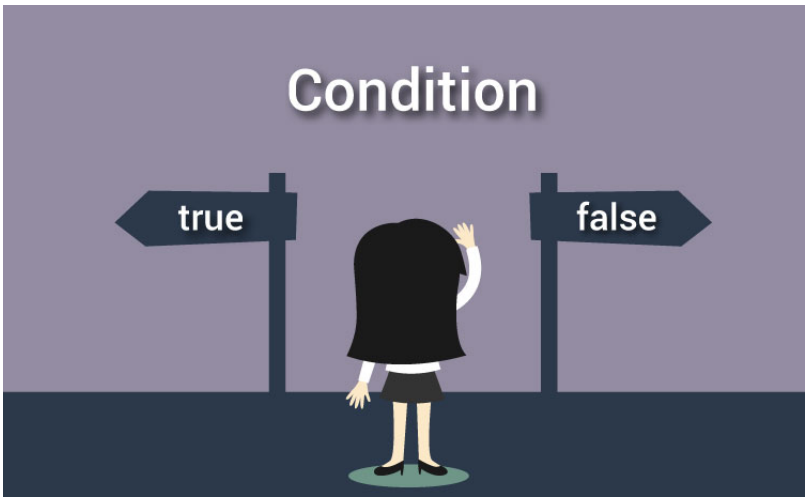
Alapvető vezérlési utasítások a programozásban

If



Ha azt szeretnénk, hogy valami akkor történjen, ha megfelel vagy épp nem felel meg valamilyen feltételnek (valami egyenlő-e valamivel, nagyobb-e, ...) akkor feltétel vizsgálatot kell végezni és az eredmény szerinti feladatot elvégezni.

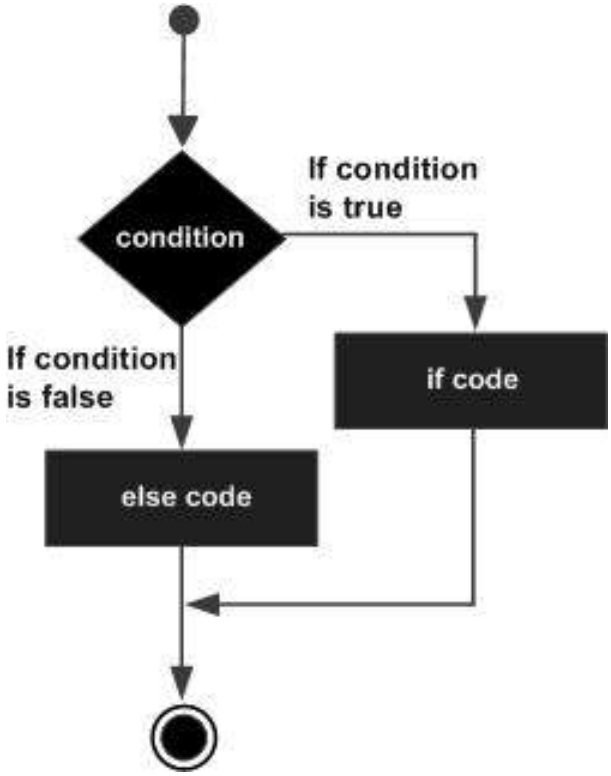
Például, ha gyököt akarunk vonni egy beadott számból szükséges lehet ilyen vizsgálat. Ugyanis ha pozitív a szám semmi gond, ha viszont negatív, akkor ezt kezelni kell, mondjuk szorozzuk meg -1-el.

Ez a kérdés magyarul hogy is szól: Ha a szám nagyobb (vagy egyenlő) mint 0 akkor vonjunk gyököt, ha pedig nem akkor szorozzuk meg mínusz eggyel és utána vonjunk gyököt belőle. Lehet érzeni a mondatban, mit mikor kell elvégezni.

Ennek a műveleteknek az elvégzésére több paranccsal is alkalmas, de a leggyakoribb az `if-else` parancs. Ez lényegében egy elágazás, ami azt teszik, hogy ha igaz az állítás, akkor megcsinál valamit, ha meg hamis, akkor mást.

Szerkezete:

```
if (feltétel vizsgálat):
   utasítás1
else:
   utasítás2
```



Megjegyzés: Nem kötelező mindkét ágban utasítást adni, ilyenkor csakis a feltétel teljesülésekor hajtódik végre utasítás.

> Tehát a következő parancs is értelmes:
>
> ```
> if "élet értelme" == 42:
>     print("Naná")  # És itt nem adunk meg else ágat
> ```

Példák

In [ ]:

```
var1 = 100
if var1:
   print("A változónak van értéke")
   print(var1)
else:
   print("A változónak nincs értéke")
   print(var1)
```

In [ ]:

```
var1 = "cica"
if var1:
   print("A változónak van értéke")
   print(var1)
else:
   print("A változónak nincs értéke")
   print(var1)
```

In [ ]:

```
var1 = ""
if var1:
   print("A változónak van értéke")
   print(var1)
else:
   print("A változónak nincs értéke")
   print(var1)
```

Adjunk magyarázatot a kovetkezőre:

In [ ]:

```
var1 = 0
if var1:
   print("A változónak van értéke")
   print(var1)
else:
   print("A változónak nincs értéke")
   print(var1)
```

Többszörös elágazások (`if-elif-else`)

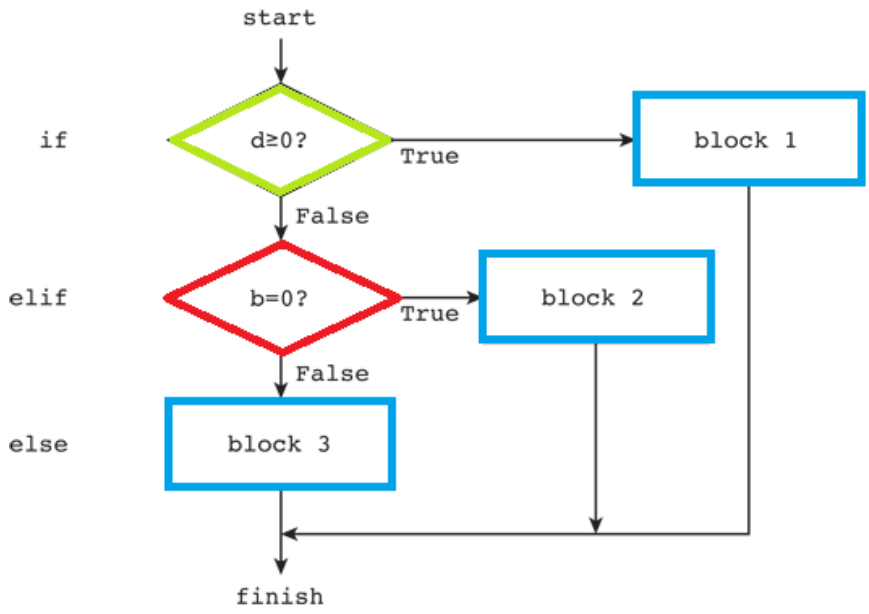
Az elágazások egymásba fűzhetőek két módon. Vagy egy már nyitott elágazásban helyezünk el újabb elágazást, vagy többszörös feltételvizsgálatot végzünk az elágazásunkon. Lássuk most az első esetet:

In [ ]:

```
szam=int(input("Kérem adjon meg egy egész számot:"))

if szam > 0:
    if szam%2 == 0:
        print("A megadott szám páros pozitív")
    else:
        print("A megadott szám páratlan pozitív")
else:
    if szam%2 == 0:
        print("A megadott szám páros negatív")
    else:
        print("A megadott szám páratlan negatív")
```

Ha nem egymásba akarunk teljesen eltérő feltételeket vizsgálni, hanem összefüggően, több feltételt külön-külön vizsgálni (több elágazásunk van), akkor használjuk az `if-elif-else` szerkezetet:



In [ ]:

```
szam=float(input("Kérem adjon meg egy számot:"))

if szam == 0:
    print("nulla")
elif szam > 0:
    print("pozitív")
else:
    print("negatív")
```

Megjegyzés: Törekedjünk a lehető legkevesebb új elágazás használatára. Mivel a sok elágazás átláthatatlanná teszi a program működését. Azaz szerencsésebb az `if-elif-else` használata, mint sok önálló `if-else`.

Oldjuk meg együtt

Írjunk egy if tömböt amely a nap és az óra valtozók megadott értékei alapján eldönti, hogy épp az adott időben a diák mit csinál. Az if tömb válasza az alábbiak szerint osztandó ki:

- A fiúk is és a lányok is hétköznap délelőtt tanulnak.
- 12-től fociznak.
- Hétvégén mindenki kirándul.
- Mindennap mindenki 8-kor megy aludni, és reggel 8 kor kel.

In [ ]:

```
# Megoldás helye
```

For

A számítógépek legfontosabb tulajdonságai közt szerepel az, hogy nagyon gyorsak és "fáradhatatlanok". Olyan feladatok megoldásában a leghatékonyabbak, amikor a feladatot kevés munkával meg lehet fogalmazni ("az alkotó pihen") de végrehajtása nagyon sok ismétlést, iterációt igényel ("a gép forog"). Az iteráció (angol: iterate) azt jelenti, hogy például egy lista elemein egyesével végigmegy a program, és műveleteket végez el rajtuk.

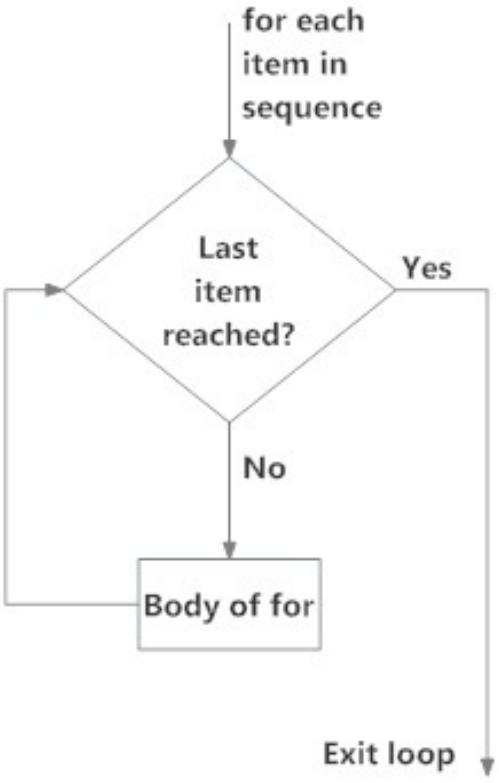


Fig: operation of for loop

A Python-ban, az egyik erre használható utasítás a **for** parancs (magyarul kb. a ...-ra, azaz pl. a hét minden napjára, a lista minden elemére):

In [ ]:

```
days_of_the_week = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Sat
urday"]
```

In [ ]:

```
for day in days_of_the_week:
    print(day)
```

Ez a kódrészlet a **days_of_the_week** listán megy végig és a meglátogatott elemet hozzárendeli a **day** változóhoz, amit *ciklusváltozónak* is neveznek. Ezek után mindent végrehajt amit a beljebb tabulált (angolul: *indented*) parancsblokkban írtunk (most csak egy **print** utasítás), amihez felhasználhatja a ciklusváltozót is. Miután vége a beljebb tabulált régiónak, kilép a ciklusból.

Szinte minden programnyelv használ hasonló ciklusokat. A C, C++, Java nyelvekben kapcsos zárójeleket {} használnak a ciklusok elkülönítésére, kb. így:

```
for (i=1..10) {print i}
```

A FORTRAN nyelvben az *END* szócska kiírása jelzi a ciklus végét. A pythonban a kettőspont (":"), majd szóközökkel beljebb írt sorok szolgálnak erre. Ha más programokból másolunk át részleteket figyeljünk arra hogy a behúzás helyett nem TAB-ot használ. (A TAB néhol megengedett de kerülendő. A modern python kódolási stílusirányzat (http://django.arek.uni-obuda.hu/python3-doc/html/tutorial/controlflow.html#intermezzo-kodolasi-stilus) minden behúzást 4 szóköznek javasol.)

Annak semmi jelentősége nincs, hogy a példában a **day** nevet adtunk az iterációban szereplő *ciklusváltozó*nak. A program semmit se tud az emberi időszámításról, például, hogy a hétben napok vannak és nem kiscicák:

In [ ]:

```
for macska in days_of_the_week:
    print(macska)
```

A ciklus utasításblokkja állhat több utasításból is:

In [ ]:

```
for day in days_of_the_week:
    statement = "Today is " + day
    print(statement)
```

A **range()** parancs remekül használható ha a **for** ciklusban adott számú műveletet szeretnénk elvégezni:

In [ ]:

```
for i in range(20):
    print("The square of ",i," is ",i*i)
```

Akkor válik mindez még érdekesebbé, ha az eddig tanult iterációt és feltétel vizsgálatot kombináljuk:

In [ ]:

```
for day in days_of_the_week:
    statement = "Today is " + day
    print(statement)
    if day == "Sunday":
        print ("   Sleep in")
    elif day == "Saturday":
        print ("   Do chores")
    else:
        print ("   Go to work")
```

Figyeljük meg a fenti példában hogy ágyazódik egymás alá a **for** és az **if**!

Egy példa program: Fibbonacci sorozat

A Fibonacci (http://en.wikipedia.org/wiki/Fibonacci_number) sorozat első két eleme 0 és 1, majd a következő elemet mindig az előző kettő összegéből számoljuk ki: 0,1,1,2,3,5,8,13,21,34,55,89,...

Ha nagyobb **n** értékekre is ki akarjuk számolni a sorozatot, ez kiváló feladat lehet egy fáradhatatlan és gyors számítógépnek!

In [ ]:

```
n=int(input("Adjon meg egy egész számot:")) #Bekér a program egy számot (ennyi tagú
 lesz a sorozat)

if n<0: # Megvizsgáljuk, hogy nem-e negatív számot adott-e meg a felhasználó
    print("Negatív számot adott meg, megszorzom -1-gyel.\n")
    n=n*(-1)

sequence = [0,1] # A két kezdőérték
for i in range(2,n): # számok 2-től n-ig, Figyelni kell hogy n ne legyen kisebb mint
 2!!
    sequence.append(sequence[i-1]+sequence[i-2]) # Így számoljuk a következő tagot
print (sequence)
```

Nézzük végig lépésről lépésre! Először **n** értékét, azaz a kiszámolandó sorozat hosszát állítjuk be 10-re. A sorozatot majdan tároló listát **sequence**-nek neveztük el, és *inicializáltuk* az első két értékkel. A "kézi munka" után következhet a gép automatikus munkája, az iteráció.

Az iterációt 2-vel kezdjük (ez ugye a 0-s indexelés miatt a 3. elem lesz, hisz az első kettőt már mi megadtuk) és **n**-ig, a megadott lista méretig számolunk.

A ciklus törzsében az addig kiszámolt lista végére hozzátűzzük (**append**) az előző két tag összegét. A ciklus vége után kiíratjuk az eredményt.

In [ ]:

```
dat=[];
for i in range(10):
    dat.append(i**2)
```

In [ ]:

```
dat
```