➢ Tests

if … : Permet de tester une condition (ou plusieurs avec « *and* » « *or* » ou « *not* ») si la condition est vérifiée (*True*), on traite les instructions situées après la tabulation

else : Permet de traiter le cas où la condition précédentes (ou les conditions) n'est pas vérifiée, on exécute alors les instructions situées après la tabulation

elif … : Permet de condenser un « *else* » avec un « *if* »

Remarque : On peut imbriquer plusieurs tests les uns dans les autres, attention à bien identifier les blocs avec les tabulations.

Exemples

```
if n != 0 :
    print(n, "est non nul")

if p%2 == 0 :
    print(p, "est pair")
else :
    print(p, "est impair")

if a > 1 :
    print(a, "est supérieur à 1")
elif a < 0 :
    print(a, "est négatif")
else :
    print(a, "est entre 0 et 1")
```

➢ Boucles inconditionnelles

for … in … : Permet de répéter les instructions situées après la tabulation pendant qu'une variable décrit un ensemble de valeurs prises dans une chaîne de caractère, une liste, un tuple ou un ensemble.
(Voir aussi les instructions *enumerate* et *zip*)

Remarque : L'instruction « *range* » permet de créer une liste virtuelle.
range(n) crée une liste virtuelle de n valeurs allant de 0 à n – 1, *range*(a, b) une liste virtuelle de a à b – 1,

Exemples

```
for i in (1, 2, 3) :
    print(i)

for c in "bonjour" :
    print(c)

for k in range(5) :
    print(k)
```

➢ Boucles conditionnelles

while … : Permet de répéter les instructions situées après la tabulation tant que la condition indiquée est vérifiée.

Attention : Il y a un risque de créer une boucle infinie si la condition n'est jamais réfutée !

Exemple

```
n = 10
while n**2 < 1000 :
    print(n, "² = ", n**2)
    n = n + 1
```

Compléments

break Permet de sortir d'une boucle « *for* » ou « *while* » et de continuer la suite des instructions du programme situées après le bloc.

continue Permet de passer directement à la boucle suivante du « *for* » ou du « *while* » en ignorant les instructions situées après dans le bloc.

➢ Exceptions

try, except : Permet d'exécuter des instructions en gérant les erreurs !

Remarque : On peut même différencier les erreurs possibles (*NameError, TypeError, ZeroDivisionError, …*)

Exemple

```
a = input()
try :
    print("inverse : ", 1/float(a))
except :
    print("erreur ! ")
```

Compléments

pass Permet de créer un bloc dans lequel rien ne se passe ! (utile dans les tests ou exceptions)