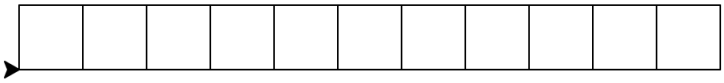
Tests



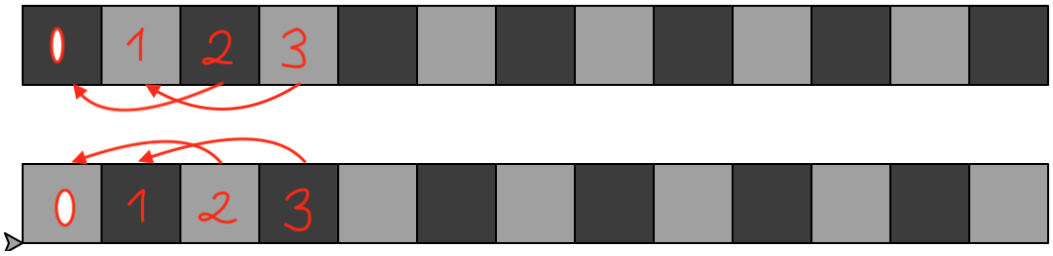
On a vu précédemment qu'une ligne de carrés était construite à l'aide d'une répétition codée par

par une boucle for

```
def dessine_carre(cote):
    for _ in range(4):
        forward(cote)
        left(90)

def dessine_ligne(n,cote):
    for _ in range(n):
        dessine_carre(cote)
        forward(cote)
    #on remet la tortue dans
    #son état initial
    pu()
    left(180)
    forward(n*cote)
    left(180)
```

On voudrait maintenant dessiner une ligne de carrés alternant les carrés gris clair et les carrés gris foncés



Cette fois ci on va utiliser un compteur de boucle `i` dans la boucle de telle sorte que `i` est le nombre de construction d'un carré (0 étant associé au carré le plus à gauche possible)

Sur le dessin ci-dessus on constate que le coloriage est **PÉRIODIQUE**, autrement dit les carrés ayant un nombre de construction `i` **pair** ont la même couleur que le premier carré de numéro 0 et les carrés ayant un nombre de construction `i` **impair** ont la même couleur que le carré de nombre 1

Pour réaliser les lignes bicolores on va donc utiliser une variable `debut` égale à 0 ou 1, et on décide que si la valeur de `debut` est 0 alors le premier carré est colorié en GRIS FONCÉ, et en GRIS CLAIR sinon

Maintenant pour tout carré de nombre `i`, si `i` a la même parité que `debut` alors on colorie le carré en GRIS FONCÉ, et en GRIS CLAIR sinon

En Python "si `i` a la même parité que `debut` " s'écrit

`if i % 2 == debut`, car `a % b` calcule le reste de la division euclidienne de `a` par `b`

Par conséquent `i % 2` est le reste de la division euclidienne par 2, donc 0 ou 1

Enfin le symbole `a == b` permet de **comparer (égalité)** les valeurs de `a` et `b`

Maintenant on va traduire la phrase "si `i` a la même parité que `debut` alors on colorie le carré en GRIS FONCÉ, et en GRIS CLAIR sinon" par

```
if i \% 2 == debut:
    fillcolor(GRIS_FONCE)
else:
    fillcolor(GRIS_CLAIR)
```

d'où la fonction `dessine_ligne_bicolore(nb_carres,cote,debut)`

```
def dessine_ligne_bicolore(nb_carres,cote,debut):
    depart = xcor(),ycor()
    for i in range(nb_carres):
        if i \% 2 == debut:
            fillcolor(GRIS_FONCE)
        else:
            fillcolor(GRIS_CLAIR)
        begin_fill()
        dessine_carre(cote)
        end_fill()
        forward(cote)
    penup()
    goto(depart)
```

A l'exécution

`dessine_ligne_bicolore(6,50,0)` va dessiner une ligne bicolore de 6 carrés commençant par un carré GRIS FONCÉ

`dessine_ligne_bicolore(6,50,1)` va dessiner une ligne bicolore de 6 carrés commençant par un carré GRIS CLAIR

Regardons un peu en détail le code

1 Test : Si .....Alors .....Sinon ...

```
def dessine_ligne_bicolore(nb_carres,cote,debut):
    depart = xcor(),ycor()
    for i in range(nb_carres):
        if i \% 2 == debut:
            fillcolor(GRIS_FONCE)
        else:
            fillcolor(GRIS_CLAIR)
        begin_fill()
        dessine_carre(cote)
        end_fill()
        forward(cote)
    penup()
    goto(depart)
```

1. Il y a plusieurs blocs, celui de la fonction, celui du bloc for et enfin celui du bloc if et du bloc else
2. Le bloc if (en vert) est exécuté uniquement si **l'expression booléenne** en rouge est **vraie**

   Autrement dit la couleur de remplissage est GRIS FONCÉ
3. **l'expression booléenne** `if i % 2 == debut` est soit **vraie** soit **fausse**

   Si elle est fausse alors c'est le bloc else (violet) qui est exécuté, autrement dit la couleur de remplissage est GRIS CLAIR
4. En conclusion si l'expression booléenne est vraie le bloc if est exécuté mais pas le bloc else, et à l'inverse si l'expression booléenne est fausse alors le bloc else est exécuté mais pas le bloc if
5. Parfois on n'utilise pas le else comme dans l'exemple suivant

   On lance un dé à six faces si le nombre est pair on peut relancer le dé et si on obtient encore un nombre pair alors on a gagné

```
def dessine_ligne_bicolore(nb_carres,cote,debut):
    depart = xcor(),ycor()
    for i in range(nb_carres):
        if i \% 2 == debut:
            fillcolor(GRIS_FONCE)
        else:
            fillcolor(GRIS_CLAIR)
        begin_fill()
        dessine_carre(cote)
        end_fill()
        forward(cote)
    penup()
    goto(depart)
```

2 Test : Si ... Alors ... Sinon Si ... Alors

Dans le cas précédent on n'évalue qu'une expression booléenne, parfois on aimerait en évaluer plusieurs comme dans le cas suivant :

Les différentes mentions possibles au bac en fonction de la moyenne générale m à l'épreuve sont

1. La moyenne m vérifie $m \geqslant 16$ dans ce cas la mention est Très Bien
2. La moyenne m vérifie $14 < m \leqslant 16$ dans ce cas la mention est Bien
3. La moyenne m vérifie $12 < m \leqslant 14$ dans ce cas la mention est Assez Bien
4. La moyenne m vérifie $10 < m \leqslant 12$ dans ce cas la mention Passable
5. La moyenne m vérifie $8 < m \leqslant 10$ dans ce cas l'élève passe un oral de rattrapage
6. La moyenne m vérifie $m < 8$ dans ce cas l'élève n'a pas le diplôme

Voici **la syntaxe du langage Python** nous permettant d'évaluer plusieurs expressions booléennes et en fonction du résultat exécuter des instructions ou pas

8.1. L'instruction `if`

L'instruction `if` est utilisée pour exécuter des instructions en fonction d'une condition :

```
if_stmt ::=  "if" assignment_expression ":" suite
             ("elif" assignment_expression ":" suite)*
             ["else" ":" suite]
```

Elle sélectionne exactement une des suites en évaluant les expressions une par une jusqu'à ce qu'une soit vraie (voir la section Opérations booléennes pour la définition de vrai et faux) ; ensuite cette suite est exécutée (et aucune autre partie de l'instruction `if` n'est exécutée ou évaluée). Si toutes les expressions sont fausses, la suite de la clause `else`, si elle existe, est exécutée.

On va donc écrire une fonction `mention_bac(moyenne)` qui retourne une chaîne de caractères désignant la mention obtenue en fonction de la moyenne obtenue

```
from random import randint
def mention_bac(moyenne):
    if moyenne >= 16:
       return "Mention Très Bien"
    elif 14 <= moyenne < 16:
        return "Mention Bien"
    elif 12 <= moyenne < 14:
        return "Mention Assez Bien"
    elif 10 <= moyenne < 12:
        return "Mention Passable"
    elif 8 <= moyenne < 10:
        return "Oral Second Tour"
    elif moyenne < 8:
        return "Refusé"

moy = randint(0,20)
print("la moyenne est ",moy," la mention est ",mention_bac(moy))
```

Quand on teste on observe que :

1. Tous les expressions booléennes vont être évaluées tant que la moyenne ne sera pas "encadrée" ainsi si la moyenne est de 6, toutes les expressions booléennes seront évaluées
2. Par contre si la note est de 13 les expressions `10 <= moyenne < 12")`, `8 <= moyenne < 10")` et `moyenne < 8")` ne seront pas évaluées et cela n'a rien avoir avec la présence du `return`

   Pour s'en assurer modifions légèrement le programme

```
from random import randint
def mention_bac(moyenne):
    if moyenne >= 16:
       ch =  "Mention Très Bien"
    elif 14 <= moyenne < 16:
       ch =  "Mention Bien"
    elif 12 <= moyenne < 14:
        ch =  "Mention Assez Bien"
    elif 10 <= moyenne < 12:
        ch =  "Mention Passable"
    elif 8 <= moyenne < 10:
        ch =  "Oral Second Tour"
    elif moyenne < 8:
        ch =  "Refusé"
    return ch

moy = randint(0,20)
print("la moyenne est ",moy," la mention est ",mention_bac(moy))
```

   Là encore on observe que (voir TP) :

   (a) Tous les expressions booléennes vont être évaluées tant que la moyenne ne sera pas "encadrée" ainsi si la moyenne est de 6, toutes les expressions booléennes seront évaluées

   (b) Par contre si la note est de 13 les expressions `10 <= moyenne < 12")`, `8 <= moyenne < 10")` et `moyenne < 8")` ne seront pas évaluées et cela n'a rien avoir avec la présence du `return`

3 Exercices

Ex 1

1. Corriger les erreurs dans le programme suivant

```
      x = 5
if x >= 2
print("Pi")
else
print("Thon")
```

2. Une fois corrigé et exécuté qu'y aura-t-il d'affiché à l'écran ?

Ex 2

1. Corriger les erreurs dans le programme suivant

```
x = 35
if x >= 40:
print(Bleu)
elif x >= 30:
print(Blanc)
elif x >= 20
print(Rouge)
```

2. Une fois corrigé et exécuté qu'y aura-t-il d'affiché à l'écran ?

Ex 3

Quelles sont les différences entre les trois programmes

Programme 1

```
x = 35
if x >= 40:
    x = 2*x
elif x >= 30:
    x = x + 2
elif x >= 20:
    x = x - 2
else:
    x = x**2
```

Programme 2

```
x = 35
if x >= 40:
    x = 2*x
elif x >= 30:
    x = x + 2
elif x >= 20:
    x = x - 2
elif x >= 10
    x = x**2
```

Programme 3

```
x = 35
if x >= 40:
    x = 2*x
else:
    if x >= 30:
        x = x + 2
    if x >= 20:
        x = x - 2
    else:
        x = x**2
```

Changer la valeur initiale de x

Ex 4

Définir une fonction Python `max(a,b)` qui retourne le plus grand des deux nombres a et b

Ex 5

Voici la règle d'un jeu de dés :

On lance quatre dés à six faces, si il y a au moins un six on gagne sinon on perd

Définir une fonction `gagne()` qui retourne Vrai si on gagne à ce jeu et faux sinon