

1 Conditions et tests

Définition 1.1 (Condition) Une *condition* est un énoncé qui peut être *vrai ou faux*.

Lorsque l'on teste si une condition est réalisée, la donnée renvoyée est `True` ou `False`, c'est-à-dire de type `bool`. On distingue deux sortes de tests : les tests avec les opérateurs de comparaison et les tests avec les opérateurs logiques.

1.1 Les opérateurs de comparaison

égal	différent	inférieur	supérieur	inférieur strict	supérieur strict
<code>==</code>	<code>!=</code>	<code><=</code>	<code>>=</code>	<code><</code>	<code>></code>

Exemple 1.2 Les instructions

```
3 == 5 ; "ab" != "cd" ; 5 <= 8 ; 4 >= -3 ; 3 < 1; "e" > "a"
```

renvoient respectivement `False`, `True`, `True`, `True`, `False` et `True`.

Remarque 1.3

- * Les opérateurs `==` et `!=` peuvent aussi s'appliquer à des listes, des tuples, etc...
- * Les opérateurs `<=`, `>=`, `<` et `>` s'appliquent bien sûr aux entiers et flottants, mais également aux chaînes de caractères. Dans ce cas, l'ordre utilisé est celui de l'ordre alphabétique.
- * Aux opérateurs de comparaison précédents s'ajoute l'opérateur `in` qui permet de tester si un élément appartient à une structure.

Exemple 1.4 L'instruction `3 in [1, 7, 3, 2]` renvoie `True`. L'instruction `'on' in "bonjour"` renvoie également `True`.

1.2 Les opérateurs logiques

ou	et	non
<code>or</code>	<code>and</code>	<code>not</code>

Exemple 1.5 Les instructions

```
3==3 or 9>24 ; 3==3 and 9>24 ; not (3==3)
```

renvoient respectivement `True`, `False` et `False`.

Remarque 1.6 On peut bien sûr combiner les opérateurs logiques entre eux, mais attention aux ordres de priorité...

Exemple 1.7 Les instructions

```
(3 < 4 or "a" < "bc") and -2 > 1; 3 < 4 or ("a" < "bc" and -2 > 1)
```

renvoient respectivement `False` et `True`.

2 Les structures d'embranchements

Définition 2.1 (Instruction conditionnelle) Une *instruction conditionnelle* ou *embranchement* permet d'effectuer une séquence d'instructions si une condition donnée est vérifiée et éventuellement une séquence d'instructions alternative si celle-ci est fausse.

2.1 La structure `si ... alors ...`

La structure algorithmique

```
si une condition est vraie alors
    bloc d'instructions
fin si
```

s'écrit

```
if condition:                # ne pas oublier les :
    bloc d'instructions      # indentation obligatoire
# pour terminer, on stoppe l'indentation
```

Exemple 2.2 Le code Python suivant permet d'afficher True si le nombre x est positif :

```
if x >= 0:
    print(True)
```

2.2 La structure si ... alors ... sinon ...

La structure algorithmique

```
si une condition est vraie alors
    bloc d'instructions 1
sinon
    bloc d'instructions 2
fin si
```

s'écrit

```
if condition:
    bloc d'instructions 1      # ne pas oublier les :
else:
    bloc d'instructions 2      # indentation obligatoire
# pour terminer, on stoppe l'indentation
```

Exemple 2.3 Le code Python suivant permet d'afficher négatif ou positif suivant le signe de x :

```
if x<=0:
    print("négatif")
else:
    print("positif")
```

2.3 Les structures d'embranchements multiples

La structure algorithmique

```
si condition 1 est vraie alors
    bloc d'instructions 1
sinon si condition 2 est vraie alors
    bloc d'instructions 2
... (on peut mettre autant de sinon si que nécessaire)
sinon
    dernier bloc d'instructions
fin si
```

s'écrit

```
if condition 1:
    bloc d'instructions 1      # ne pas oublier les :
elif condition 2:
    bloc d'instructions 2      # indentation obligatoire
...
else:
    dernier bloc d'instructions # ne pas oublier les :
# pour terminer, on stoppe l'indentation
```

Exemple 2.4 Le code Python suivant permet d'afficher le signe de x :

```
if x<0:
    print("strictement négatif")
elif x==0 :
    print("nul")
else :
    print("strictement positif")
```

3 Les boucles

3.1 Les structures de boucles itératives ou boucles bornées

Définition 3.1 Une *boucle itérative* permet de répéter une séquence d'instructions un nombre fixé de fois.

Voici la syntaxe pour répéter n fois une séquence d'instructions où n désigne un entier naturel fixé.

Algorithme

```
pour i allant de 0 à n-1
    séquence d'instructions
fin pour
```

Programmation Python

```
for i in range(n) :
    séquence d'instructions
```

Attention à ne pas oublier les : et l'indentation.

Exemple 3.2 Une somme de 2000 € est placée à 2%. Le code Python suivant affiche la somme disponible au bout de 20 ans :

```
S = 2000          # on initialise la somme S
for i in range(20):
    S = S * 1.02   # on calcule l'évolution des sommes S au cours des 20 ans
print(S)
```

Définition 3.3 Les données de type `range` permettent d'énumérer une liste ordonnée de nombres entiers. Pour définir une donnée de ce type, on utilise la fonction `range()` qui admet de un à trois paramètres :

- ★ `range(n)` : cette fonction renvoie une structure constituée de tous les nombres entiers compris entre 0 et $n - 1$; le paramètre n est un nombre entier. Si $n \leq 0$, la structure est vide.
- ★ `range(m, n)` : cette fonction renvoie une structure constituée de tous les nombres entiers compris entre m et $n - 1$; les paramètres m et n sont des nombres entiers. Si $n \leq m$, la structure est vide.
- ★ `range(m, n, p)` : cette fonction renvoie une structure constituée de tous les nombres entiers compris entre m et $|n| - 1$ avec un pas de p ; les paramètres m , n et p sont des nombres entiers et p est non nul.
 - Si $n \leq m$ et $p > 0$ ou si $m \leq n$ et $p < 0$, la structure est vide.
 - Si $m < n$ et $p > 0$, la structure est constituée de nombres entiers croissants.
 - Si $n < m$ et $p < 0$, la structure est constituée de nombres entiers décroissants.

Exemple 3.4 `range(1, 11, 2)` est constitué des entiers impairs (le pas est 2) de 1 à 10 soit 1,3,5,7,9.

Remarque 3.5 Nous verrons plus tard dans le cours que des boucles itératives peuvent être construites à partir de n'importe quelle structure itérable (listes, chaînes de caractères, tuples, dictionnaires...).

3.2 Les structures de boucles conditionnelles ou boucles non bornées

Définition 3.6 Une *boucle conditionnelle* permet de répéter une séquence d'instruction tant qu'une condition est vérifiée.

Algorithme

```
tant que condition vraie
    séquence d'instructions
fin tant que
```

Programmation Python

```
while condition vraie:
    séquence d'instructions
```

Attention à ne pas oublier les : et l'indentation.

Exemple 3.7 Une somme de 2000 € est placée à 2%. Le code Python suivant affiche au bout de combien d'années on dispose d'une somme d'au moins 5000 € :

```
S = 2000          # on initialise la somme S
n = 0             # on initialise le compteur « manuel » n
while S < 5000:
    S = S * 1.02   # on calcule la nouvelle somme disponible chaque année
    n = n + 1      # on incrémente le compteur d'années de 1
print(n)
```

4 La bibliothèque turtle

Ce paragraphe présente les différentes fonctionnalités de la bibliothèque `turtle`. Celle-ci permet de réaliser des dessins géométriques correspondant à la trace d'une petite « tortue » virtuelle (représentée par une petite flèche) se déplaçant dans une fenêtre graphique. Le déplacement de la tortue sur l'écran est contrôlé par des instructions simples issues de la bibliothèque `turtle`. Pour charger cette bibliothèque, n'oubliez pas de commencer votre script par l'instruction

```
from turtle import *
```

ou

```
import turtle
```

en précédant les appels aux fonctions de la bibliothèque `Turtle` par `turtle..`

4.1 Manipulation de la fenêtre graphique

- * `setup(W, H)` : initialise la fenêtre graphique ; ses deux arguments représentent respectivement sa largeur et sa hauteur. L'origine (0;0) est situé au centre de la fenêtre et la tortue est placée initialement à cet endroit avec une orientation vers la droite.
- * `window_width()` et `window_height()` : renvoient respectivement la largeur et la hauteur de la fenêtre graphique.
- * `bgcolor(c)` : donne la couleur `c` à l'arrière-plan de la fenêtre graphique ; la couleur `c` est une couleur prédéfinie donnée sous forme de chaîne de caractères (voir tableau ci-dessous). On peut également donner les couleurs sous la forme d'un triplet `(r, g, b)` en indiquant la commande `colormode(255)`.
- * `reset()` : efface le contenu de la fenêtre graphique et replace la tortue en son centre.
- * `clear()` : efface le contenu de la fenêtre graphique mais conserve la tortue en sa dernière position.
- * `title(name)` : donne comme nom à la fenêtre graphique la chaîne de caractères `name`.
- * `exitonclick()` : quitte la fenêtre graphique en cliquant dedans.
- * `getcanvas().postscript(file="monoeuvre.eps")` : sauvegarde le dessin obtenu dans un fichier `.eps` (ici, `monoeuvre`), de façon à pouvoir l'éditer ultérieurement, par exemple avec HTML.

4.2 Gestion de la tortue

- * `goto(x, y)` : amène la tortue au point de coordonnées `(x; y)`.
- * `forward(d)` ou `fd(d)` : avance la tortue d'une distance `d` (par rapport à l'orientation).
- * `backward(d)` ou `bk(d)` : recule la tortue d'une distance `d` (par rapport à son orientation).
- * `position()` ou `pos()` : renvoie la position de la tortue.
- * `home()` : replace la tortue au centre de la fenêtre graphique.
- * `distance(x, y)` : renvoie la distance entre la tortue et le point de coordonnées `(x; y)`.
- * `left(a)` ou `lt(a)` : pivote la tortue vers la gauche d'un angle `a` (exprimé en degrés).
- * `right(a)` ou `rt(a)` : pivote la tortue vers la droite d'un angle `a` (exprimé en degrés).
- * `setheading(a)` : oriente la tortue d'un angle `a` (en degrés) ; en particulier, si `a=0`, la tortue est orientée vers la droite, si `a=90`, elle est orientée vers le haut, si `a=180`, elle est orientée vers la gauche et si `a=270`, elle est orientée vers le bas.
- * `heading()` : renvoie l'orientation de la tortue.
- * `hideturtle()` ou `ht()` : rend la tortue invisible (accélère le tracé).
- * `showturtle()` ou `st()` : rend la tortue visible.

4.3 Tracé de formes

- * `up()` ou `penup()` ou `pu()` : relève la tortue (pour pouvoir se déplacer sans dessiner).
- * `down()` ou `pendown()` ou `pd()` : abaisse la tortue (pour recommencer à dessiner).
- * `circle(r)` : trace un cercle de rayon `r` ; le tracé commence à l'endroit où est la tortue, dans la direction de la tortue puis tourne sur la gauche.
- * `circle(r, a)` : trace un arc de cercle de rayon `r` et d'angle `a` (exprimé en degrés).
- * `color(c)` : donne la couleur `c` au trait.
- * `color(c, d)` : donne la couleur `c` au trait et remplit la figure avec la couleur `d`.

- ★ `begin_fill()` ... `end_fill()` : remplit un contour à l'aide de la couleur sélectionnée; les ... doivent être remplacés par une suite d'instructions définissant une forme géométrique.
- ★ `width(e)` : donne l'épaisseur `e` au trait.
- ★ `shape(f)` : change la forme de la tortue, `f` est une chaîne de caractères ("turtle", "circle", "arrow" (par défaut), "square", "triangle", "classic", "blank")
- ★ `speed(v)` : règle la vitesse `v` de tracé ("slowest", "slow", "normal" (par défaut), "fast", "fastest").
- ★ `write(text)` : écrit la chaîne de caractères `text` dans la fenêtre graphique.

4.4 Couleurs prédéfinies

Les couleurs prédéfinies sont données sous la forme de chaîne de caractères. Par exemple, "black" donne la couleur noire et "white" la couleur blanche. Pour les autres couleurs, on a le tableau de correspondance suivant :

bleu	rouge	vert	jaune	marron	rose	orange	violet	gris
"blue"	"red"	"green"	"yellow"	"brown"	"pink"	"orange"	"purple"	"grey"