Chapter 1

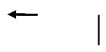
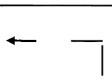
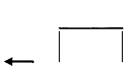
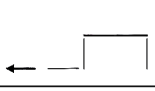
Méthode de tracé simple

1.1 Méthode OEDR: principe

Le codage OEDR (odd-even-drawing-rule) est une méthode de tracé des suites de nombres composées de 0 et de 1 exclusivement. On parcourt la suite, chaque chiffre est représenté par un segment :

- Si le chiffre est 1, on poursuit tout droit.
- Si le chiffre est 0, on poursuit tout droit et on tourne de 90° à droite si le rang du 0 est pair et à gauche s'il est impair.

Exemple: Le tracé de la suite {01001} est expliqué ci-dessous:

Rang	Valeur	Protocole	Tracé
1	0	On trace le segment et on tourne de 90° à gauche (0 est le terme de rang 1, impair).	
2	1	On trace le segment et on continue tout droit.	
3	0	On trace le segment et on tourne de 90° à gauche (rang impair) en suivant la courbe.	
4	0	On trace le segment et on tourne de 90° à droite (rang pair) en suivant la courbe.	
5	1	On trace le segment et on continue tout droit.	

Exemple: Tracer à la main la suite: {000010000}

1.2 Algorithme

Ecrire un algorithme `oedr()` : celui-ci prend en paramètre une variable `sequence` de type `list` et affiche le tracé OEDR correspondant à l'aide du module *turtle*.

Exemple: On considère les indices de la liste, en commençant donc à l'indice 0.

```
>>>oedr([1,0,0,0,1,0,1])
```

Nous pouvons donc maintenant utiliser le tracé OEDR pour tracer des séquences prises dans l'aphabet $\{0, 1\}$.

1.3 Chaine de caractère en OEDR

On souhaite alors jouer avec les chaînes de caractère, en les représentant avec ce protocole OEDR.

Comment alors représenter une chaîne de caractère en série de 0 et de 1 pour la représenter avec la méthode OEDR ? Les caractères, comme tout le reste, doivent être transformés en séries de bits pour être lus par le processeur. On utilise pour cela un encodage. II y a longtemps, le codage d'un caractère se faisait uniquement sur 7 bits et seulement en caractères latins de base: pas de caractères accentués donc. On s'inquiétait surtout de pouvoir coder des textes en anglais, essentiellement techniques. Le code le plus courant sur les PC, et qui finit par s'imposer, fut le code **ASCII**, mais il y en avait de nombreux autres, concurrents: un même codage ou série de bits pouvait correspondre à divers caractères selon l'encodage. Par exemple, la norme **ISO-8859-1 (latin-1)** codifie tous les caractères accentués du français ou de l'allemand. Ce qui posait évidemment problème si on ouvrait un fichier avec le mauvais encodage.

Dorénavant, le codage des caractères est autorisé sur plusieurs octets avec **Unicode**: celui-ci correspond à plusieurs codages dont le plus usuel est **UTF-8**. C'est aujourd'hui le code à utiliser ; les autres doivent être considérés comme obsolètes. Le code **UTF-8** reprend le code **ASCII** pour les caractères de base ; les autres caractères étant codés sur plusieurs octets.

Vous savez que Python est pourvu d'un type de données *chaîne de caractères* (le type *string*) qui respecte scrupuleusement la norme **Unicode**.

On imagine le jeu suivant: si vous parvenez à transformer une chaine de caractère (comme vos nom et prénom, ou ce que vous voudrez) en sa représentation ASCII binaire, vous pourrez la représenter avec un tracé OEDR.

A faire: Ecrivez un code que vous factorisez au mieux permettant de prendre en paramètre une chaîne de caractère et renvoyant les valeurs binaires correspondantes pour effectuer un tracé OEDR. Vous pouvez utiliser *une* fonction *built-in* de Python.

1.4 Avec de la couleur

Créer une liste de couleur, appeler une couleur choisie au hasard dans cette liste à chaque pas. **Amélioration possible**: appeler une couleur dépendant du nombre d'itération.

1.5 La suite des mots de Fibonacci

1.5.1 Suite de Fibonacci

On rappelle que la suite de *Fibonacci* est définie par la relation de récurrence:

$$u_1{=}1,\ u_2 = 0 \text{ et } u_n = u_{n-1} + u_{n-2}$$

On crée la suite des **mots de Fibonacci** à partir de cette suite que nous représenterons.

1.5.2 Suite des mots de Fibonacci

Il y a plusieurs façon de procéder:

1. Par concaténation.

   - On remplace l'addition par la concaténation notée *.
   - Le premier mot ou terme est m_1=1, le second est m_2=0.
   - Les suivants sont définies par $m_n = m_{n-1} * m_{n-2}$.

   On a donc $m_1 = 1$ et $m_2 = 0$ d'où:

   $$m3 = m_2 * m_1 = 0 * 1 = 01 \text{ et } m_4 = m3 * m2 = 01 * 0 = 010.$$

   En Python, on pourra représenter les mots par des listes et les concaténer.

2. Par substitution.

   On pose toujours $m_1 = 1$ et $m_2 = 0$, et on substitue ces deux "lettres" de la façon suivante:

   $$1 \longrightarrow 0 \text{ et } 0 \longrightarrow 01$$

   $m_3 = 01$, $m_4 = 010$ et ainsi de suite.

A faire

1. Tester quelques **mots de Fibonacci**, par exemple, m_5, m_6, m_7 puis les représenter à l'aide de votre fonction `oedr()`.
2. Ecrire une fonction `mot_fibo(m1, m2, n)` récursive renvoyant le mot de Fibonacci de rang n.

```
>>>mot_fibo([1],[0], 5)
[0, 1, 0, 0, 1]
```

1.5.3 Propriétés de la suite

Ecrire un algorithme qui compte le nombre d'éléments de chacun des mots, puis le nombre de 0 et le nombre de 1 de chacun de ces termes, par exemple jusqu'au rang 8. Regrouper ces résultats dans le tableau suivant. Que remarquez-vous ?

Mot m_n	Nombres de termes	Nombre de 1	Nombre de 0
1	1	1	0
0	1	0	1
01			

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]