Module 3 : Structures de données

Situations

Caractères & Codes

On donne le programme suivant :

```python
num = 69
car = "E"
print("Caractère :", chr(num))
print("Code :", ord(car))
```

1. Copier/coller, puis tester le programme. Qu'est ce qu'il affiche ?
2. Modifier les valeurs de `num` et de `car` :
   - `num = 101` et `car = "e"`
   - `num = 52` et `car = "4"`
   - `num = 90` et `car = "Z"`
   - `num = 33` et `car = "!"`
3. En déduire le rôle des fonctions `chr(num)` et `ord(car)`.
4. En se référant à la table ASCII suivante :

   Trouver quels sont les caractères dont le code est :
   - 35
   - 62
   - 59

   Trouver quels sont les codes des caractères :
   - "+"
   - ")"
   - "z"

Nombre automorphe

Un **nombre automorphe** est un entier naturel dont la suite des chiffres du carré se termine par celle du nombre lui-même. Par exemple,

$$5^2 = 25,\ 6^2 = 36,\ 25^2 = 625,$$

$$76^2 = 5776,\ 890625^2 = 793212890625$$

On demande d'écrire un programme qui teste si un nombre est automorphe. On donne, pour celà, le programme incomplet suivant :

```python
# Todo 1 : Enlever le commentaire # de la ligne suivante
# n = int(input("Donner un nombre > 0 ? "))
# Todo 2 : à supprimer
n = 90625
# Todo 3 : calculer le carré de n
nc = 8212890625
# Todo 4 : convertir n et nc en chaînes de caractères
nch = "90625"
ncch = "8212890625"
# Todo 5 : trouver la longueur de nch et de ncch
l1 = 5
l2 = 10
# Todo 6 : utiliser l1 et l2 pour retrouver les l1 derniers caractères de ncch
dcar = "90625"
# Todo 7 : déterminer si n est automorphe
automorphe = True
print(n, "est automorphe ?", automorphe)
```

On demande de :

1. Déterminer, d'après **la définition d'un nombre automorphe**, quels sont les nombres automorphes dans la liste suivante :
   - 376
   - 9376
   - 17
   - 125
2. Calculer le carré de `n`, **Todo 3**, puis afficher la valeur de `nc`.

```python
print(n, "^ 2 =", nc)
```

3. Convertir `n` et `nc` en chaînes, **Todo 4**, et afficher leurs types.

```python
print(nch, "est de type", type(nch))
print(ncch, "est de type", type(ncch))
```

4. Trouver les longueurs de `nch` et de `ncch`, **Todo 5**, puis afficher le nombre de chiffres de `n` et de `nc`.

```python
print(n, "contient", l1, "chiffres")
print(nc, "contient", l2, "chiffres")
```

5. Afficher le dernier chiffre de `ncch`.

```python
print("Le dernier chiffre de", ncch, "est", "???") # remplacer "???" par le code adéquat
```

6. Afficher les deux derniers chiffres de `ncch`.

```python
print("Les deux derniers chiffres de", ncch, "sont", "???") # remplacer "???" par le code adéquat
```

7. Retrouver les `l1` derniers chiffres de `ncch`, **Todo 6**.

```python
print("Les", l1, "derniers chiffres de", ncch, "sont", "???") # remplacer "???" par le code adéquat
```

8. Afficher le premier chiffre de `nch` ?

```python
print("Le premier chiffre de", ncch, "est", "???") # remplacer "???" par le code adéquat
```

9. Afficher les deux premiers chiffres de `nch` ?

```python
print("Les deux premiers chiffres de", ncch, "est", "???") # remplacer "???" par le code adéquat
```

10. Tester si le nombre **7** est automorphe ? Qu'affiche le programme ?
11. Tester si le nombre **36** est automorphe ? Qu'affiche le programme ? Quel est le problème ?
12. Répondre à **Todo 7**.
13. Compléter le programme **Todo 2**, puis **Todo 1**.

Le plus âgé

On veut faire un programme qui détermine si une personne est plus âgée qu'une autre. Pour cela on donne le programme ci-dessous :

Python

```
nom1 = input("Nom de la 1ère personne ? ")
age1 = input("Âge de " + nom1 + " ? ")

nom2 = input("Nom de la 2ème personne ? ")
age2 = input("Âge de " + nom2 + " ? ")

c1 = age1 > age2

print(nom1, "est plus âgé que", nom2, "?", c1)
```

On demande de :

1. Dresser **le tableau de déclaration des objets (TDO)** du programme.

Objet	Type
nom1	
nom2	
age1	
age2	
c1	

2. Evaluer l'expression `"180" + "10"`.

Python

```
print("180" + "10")
```

3. Evaluer l'expression `"Âge de " + nom + " ? "` pour `nom = "Faten"`.

Python

```
nom = "Faten"
age = 16
print("Âge de " + nom + " ? ")
```

4. Déduire le rôle de l'opérateur `+`.
5. Ajouter l'instruction suivante :

Python

```
print(nom + " a " + age + " ans")
```

   Que remarquez-vous ? Quelle est la raison de cette erreur ? Comment la corriger ?
6. Tester le programme avec les valeurs suivantes, puis remplir la colonne `c1` :

nom1	age1	nom2	age2	c1
Sami	28	Safouene	20	
Abderrazek	9	Hala	3	
Emna	73	Abderrazek	9	
Tortue	150	Salim	45	

7. Déduire si le programme est correct ? Quel est le problème ? Comment le corriger ?

Pomme ou pomme de terre ?

Pour son cours de français, Ahmed a rédigé le paragraphe suivant, un peu distrait par son petit frère il a commis deux types d'erreurs :

- Il a ajouté des mots qui devront être ~~supprimés~~ pour rectifier son texte.
- Il a oublié des mots qui devront être ajoutés pour enrichir son texte.

Les bienfaits de la pomme ~~de terre~~

La pomme ~~de terre~~ est un fruit succulent qui apporte une grande satiété, se transporte facilement et peut se consommer partout.
La pomme ~~de terre~~ est particulièrement intéressante pour les sportifs : avant l'effort elle apporte de l'énergie, pendant l'effort, elle apporte des minéraux ~~des minéraux~~ et des vitamines qui rechargent l'organisme et après l'effort, elle réhydrate !

Voulant montrer ses nouveaux acquis en programmation Python, Ahmed a écris le programme suivant. Dans ce qui suit on doit l'aider à corriger son paragrpahe sans modifier directement le texte qui est déjà rédigé.

Python

```
titre = "Les bienfaits de la pomme de terre"
par1 = """La pomme de terre est un fruit qui apporte une grande satiété,
se transporte facilement et peut se consommer partout."""
par2 = """La pomme de terre est particulièrement intéressante :
- avant l'effort elle apporte de l'énergie,
- pendant l'effort, elle apporte des minéraux des minéraux et des vitamines qui rechargent l'organisme
- et après l'effort, elle réhydrate !"""

# Todo 1 : supprimer l'expression " de terre" du titre
# Todo 2 : supprimer l'expression " de terre" du par1
# Todo 3 : supprimer l'expression " de terre" du par2
# Todo 4 : supprimer l'expression " des minéraux" du par2
# Todo 5 : Insérer le mot "succulent" dans par1
# Todo 6 : Insérer l'expression "pour les sportifs" dans par2

print(titre)
print()
print(par1)
print()
print(par2)
```

On demande de :

1. Calculer et d'afficher la longueur des trois chaînes `titre`, `par1` et `par2`.

   Python

   ```
   ltit = len(titre)
   print("Le titre contient " + ltit + " caractères") # Corriger l'erreur
   ```

2. Afficher le premier caractère du premier mot de `titre`. Afficher le premier caractère du troisième mot du `titre`. Utiliser l'indexation positive, puis l'indexation négative.

```python
# Utiliser l'indexation positive
print("Le 1er caractère du titre est :")
print("Le 1er caractère du 3ème mot du titre est :")

# Utiliser l'indexation négative
print("Le 1er caractère du titre est :")
print("Le 1er caractère du 3ème mot du titre est :")
```

3. Afficher le deuxième mot du `titre` en utilisant l'indexation positive et puis l'indexation négative.

```python
# Utiliser l'indexation positive
print("Le 2ème mot du titre est :")

# Utiliser l'indexation négative
print("Le 2ème mot du titre est :")
```

4. Afficher si le `titre` contient le mot **"apporte"**, faire de même pour `par1` et `par2`.

```python
mot = "apporte"
ex_titre = True # à compléter
ex_par1 = True # à compléter
ex_par2 = False # à compléter
print("Le mot", mot, "existe dans le titre :", ex_titre)
print("Le mot", mot, "existe dans le 1er paragrpahe :", ex_par1)
print("Le mot", mot, "existe dans le 2ème paragraphe :", ex_par2)
```

5. Trouver et afficher la position du mot **"pomme"** dans chacune des variables `titre`, `par1` et `par2`.

```python
mot = "pomme"
p_tit = -1 # à compléter
p_p1 = -1 # à compléter
p_p2 = -1 # à compléter
print("Le mot", mot, "est à la position", p_tit, "dans le titre")
print("Le mot", mot, "est à la position", p_p1, "dans le 1er paragraphe")
print("Le mot", mot, "est à la position", p_p2, "dans le 2nd paragraphe")
```

6. Trouver la position du mot **" de terre"**, la longueur de ce mot et puis le supprimer des trois variables : Todo 1, Todo 2, Todo 3 et Todo 4.

```python
mot = " de terre"
l_mot = 0 # à compléter
# Todo 1 : supprimer l'expression " de terre" du titre
p_mot = -1 # à compléter
titre = titre # à compléter
```

7. Aider Ahmed à ajouter le mot **"succulent"** : Todo 5.
   - Quel est le mot qui le précède dans le titre ?
   - Chercher la position de ce mot (le mot qui vient tout juste avant le mot "succulent") dans le `par1`.
   - Ajouter ce mot dans `par1`.
8. Répondre à Todo 6.

Le Halloween & les Bonbons

A l'occasion du Halloween, Samer et ses deux soeurs aînées Sarra at Samar se déguisent pour le Halloween. Le soir, ils visitent les maisons du quartier tout en chantant. Les voisins, très reconnaissants pour le geste, remplissent leur citrouille de bonbons et de friandises.
À la fin du soirée, les trois enfants à la maison, partagent leur butin. Chaque enfant doit avoir le même nombre de bonbons.
Le reste est partagé entre la mère et le père.

- S'il reste un bonbon, il sera mangé par le père.
- S'il reste deux bonbons, l'un sera mangé par le père et l'autre par la mère.
- S'il en reste trois, deux mangés par le père et un seul par la mère.

Pour une distrubtion équitable des friandises leur père a écrit le programme suivant :

Python

```
nb = 15 # Todo 1 : Nombre total de bonbons
nbe = 3 # Todo 2 : Nombre d'enfants

nbpe = 5 # Todo 2 : Nombre de bonbons par enfant
nbr = 0 # Todo 3 : Nombre de bonbons restants

pmb = False # Todo 4 : Le père mange un bonbon ?
mmb = False # Todo 5 : La mère mange un bonbon ?

nbmp = 0 # Todo 6 : Nombre de bonbons mangé par le père
nbmm = 0 # Todo 7 : Nombre de bonbons mangé par la mère

print("Nbre de bonbons par enfant :", nbpe)
print("Nbre de bonbons restants :", nbr)
print("Le père mange des bonbons ?", pmb)
print("La mère mange des bonbons ?", mmb)
print("Nombre de bonbons mangés par le père :", nbmp)
print("Nombre de bonbons mangés par la mère :", nbmm)
```

On demande de :

1. Dresser le TDO du programme.

Objet	Type
nb	
nbe	
nbpe	
nbr	
pmb	
mmb	
nbmp	
nbmm	

2. Dresser le tableau suivant, remarquer que le nombre d'enfants est maintenant 4 au lieu de 3 :

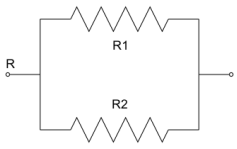
nb	nbe	nbpe	nbr	pmb	mmb	nbmp	nbmm
20	3						
21	3						
22	3						
23	3						
24	3						
25	3						

3. Calculer, dans le tableau précédent, le nombre de bonbons par enfants `nbpe`, puis le nombre de bonbons restants `nbr`. Quel est le nombre maximal de bonbons restants ?
4. Dresser le tableau suivant :

nb	nbe	nbpe	nbr	pmb	mmb	nbmp	nbmm
20	4						
21	4						
22	4						
23	4						
24	4						
25	4						

5. Refaire les calculs de la question n°3 pour les colonnes `nbpe` et `nbr`.
6. Trouver le nombre de bonbons `nb` qu'il faut collecter pour obtenir un reste égal à 5 lorsque le nombre d'enfants `nbe` est égal à 2.
7. Remplir les colonnes `pmb` (Le père mange des bonbons) et `mmb` (La mère mange des bonbons) du tableau en fonction de `nbr` (nombre de bonbons restants).
8. Remplir les colonnes `nbmp` (Bonbons pour père) et `nbmm` (Bonbons pour mère) du tableau en fonction de `nbr` (nombre de bonbons restants).
9. Trouver l'expression en Python, **Todo 2**, qui permet de calculer le nombre de bonbons par enfant.
10. Trouver l'expression en Python, **Todo 3**, qui permet de calculer le nombre de bonbons restants.
11. Trouver l'expression en Python, **Todo 4**, qui permet de déterminer s'il reste des bonbons pour le père ou non.
12. Trouver l'expression en Python, **Todo 5**, qui permet de déterminer s'il reste des bonbons pour le mère ou non.
13. Trouver l'expression en Python, **Todo 6**, qui permet de calculer le nombre de bonbons pour le père.
14. Trouver l'expression en Python, **Todo 7**, qui permet de calculer le nombre de bonbons pour la mère.
15. Saisir le nombre total de bonbons `nb` et le nombre d'enfants `nbe`, **Todo 1**.

Résistance équivalente



$$R_{eq} = \frac{R_1 R_2}{R_1 + R_2}$$

Figure 1, Résistance équivalente de deux résistances montés en parallèle

Youssef est un électronicien amateur il prend des cours d'électronique en ligne. Aujourd'hui son cours porte sur le montage de deux résistances en parallèle.

Afin de simplifier le calcul de la résistance équivalente il a écrit le programme Python suivant :

Python

```
# Todo 1 : Saisir la valeur des résistances
r1 = 100
r2 = 100

# Todo 2 : Calculer la résistance équivalente
req = 50

print("R1 =", r1, "Ω")
print("R2 =", r2, "Ω")

# Todo 3 : Afficher Req avec seulement 2 décimales
print("Req =", req, "Ω")
```

Bien que le programme précédent permet de calculer correctement la résistance équivalente de deux résistances de 100Ω chacune, il reste inachecvé et on veut :

1. Calculer la résistance équivalente de deux résistances dans les cas suivants :

r1 (Ω)	**r2 (Ω)**	**R_{eq} (Ω)**
0.1	0.2	
470	220	
1000	22000	
4700	10000	

2. Dresser le TDO du programme.

Objet	**Type**
r1	
r2	
req	

3. Permettre à l'utilisateur de saisir les valeurs des résistances par lui même, Todo 1.
4. Calculer la résistance équivalente `req` en fonction de `r1` et de `r2`, Todo 2.

   Python

   ```
   req = r1 * r2 / r1 + r2
   ```

5. Tester le programme avec les valeurs du tableau précédent. Est-ce que le programme donne des valeurs correctes ? Pourquoi ?
6. Tester le programme avec les valeurs `r1 = 100` et `r2 = 200`. Le programme doit retourner la valeur `Req = 66.66666666666667 Ω`. Limiter le nombre de déciamles à 2, Todo 3.

Logic gates

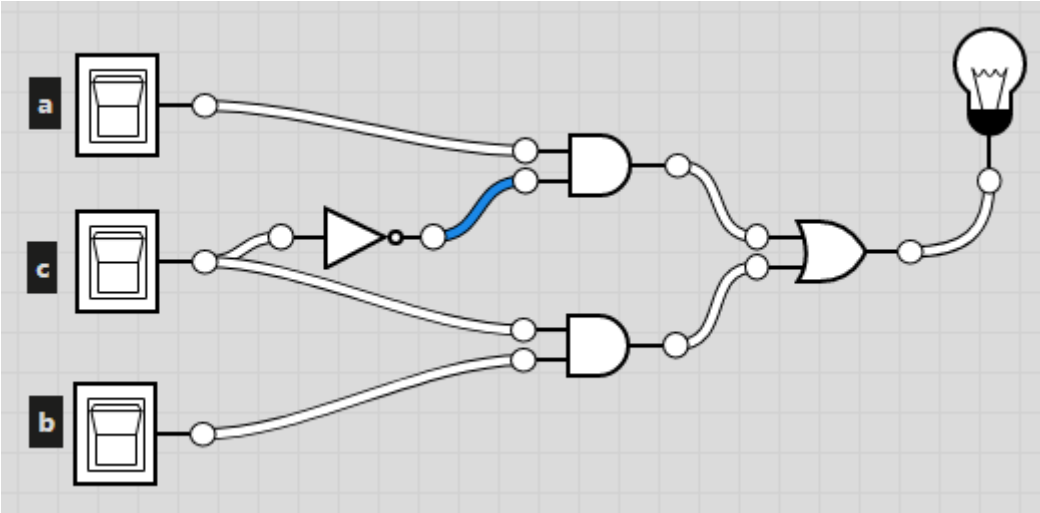


Figure 2, Portes logiques

1. Soit le circuit logique ci-dessus, on demande d'écrire l'équation logique de la lampe `L` en fonction des entrées logiques `a`, `b` et `c`.
2. Dresser le tableau de vérité de ce circuit. Puis, remplir la colonne L.

a	b	c	L
False	False	False	
False	False	True	
False	True	False	
False	True	True	
True	False	False	
True	False	True	
True	True	False	
True	True	True	

3. On donne le programme incomplet suivant qui permet de simuler ce circuit :

Python

```
# Todo 1 : Accepter la saisie de l'utilisateur
a = False
b = False
c = False

# Todo 2 : Equation logique de L
L = False

# Résultat
print(a, ",", b, ",", c, "=>", L)
```

Tester le programme.

4. Dresser le TDO du programme.

Objet	Type
a	
b	
c	
L	

5. Quel est le résultat de l'expression suivante si l'utilisateur saisit :

```python
print(input("Êtes-vous d'accord O/N ? ") == "O")
```

   - "O"
   - "o"
   - "x"

6. Changer l'expression précédent pour qu'elle retourne `True` dans le cas où l'utilisateur saisit la lettre "O" indépendamment de sa casse (Majuscule ou minuscule).
7. On suppose que les lettres **"t"** ou **"T"** signifient la valeur **True**, que les lettres **"f"** ou **"F"** signifient la valeur **False**. Et, on demande de compléter **Todo 1**.
8. Ecrire l'équation logique du circuit en utilisant les opérateurs logiques de Python, **Todo 2**.
9. Tester le bon fonctionnement du programme.

Résumé

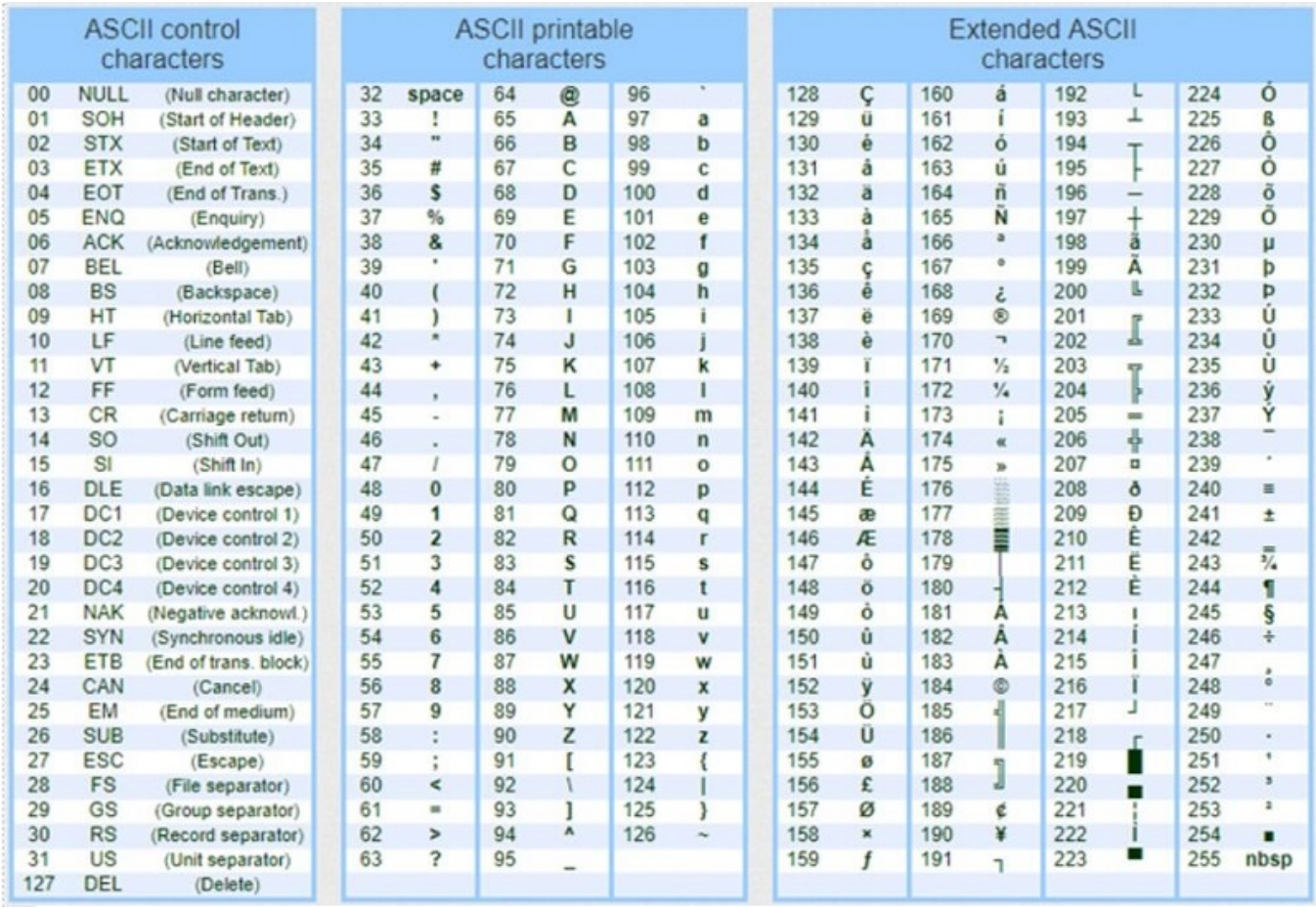
Type Caractère

Un caractère peut-être :

- Une lettre alphabétique majuscule : "A", "B", ..., "Z"
- Une lettre alphabétique minuscule : "a", "b", ..., "z"
- Un chiffre : "0", "1", ..., "9"
- Un symbole : " ", "!", "\"", "#", etc.

Chaque caractère possède un code normalisé appelé :

- **Code ASCII** pour les 256 premiers caractères (voir table ASCII)
- **Code Unicode** au dela des 256 premiers caractères.



ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	▒	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Figure 3, Table ASCII

Fonction	Rôle	Exemple

ord(car)	Retrouver le code ASCII/Unicode d'un caractère.	Python
		`car = "A"`
		`num = ord(car) # num = 65`

chr(num)	Retrouver le caractère correspondant à un code ASCII/Unicode.	Python
		`num = 64`
		`car = chr(num) # car = "@"`

Type chaîne de caractères

Une chaîne de caractères est formée par un ensemble de caractères.

Longueur

Une chaîne possède une **longueur** qui indique le nombre de ses caractères. On utilise la fonction `len(chaine)` pour déterminer la longueur d'une chaîne.
Une chaîne de longueur zéro est appelée **chaîne vide**. Elle est notée `""` ou `''` et `len("") → 0`.

Concaténation

Une chaîne peut être le résultat de concaténation de deux ou plusieurs chaînes `"Sa" + "mi" → "Sami"`.

Indexation

On peut **accéder aux caractères individuels** d'une chaîne à travers l'opérateur `[]`. Une chaîne peut-être **indexée** de deux façons différentes.

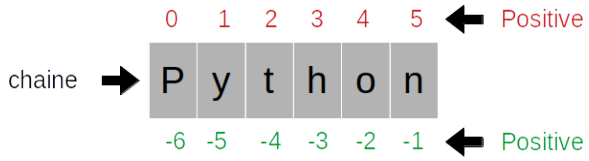


Figure 4, Indexation d'une chaîne de caractères

D'après la figure précédente, on peut retrouver le premier caractère de la `chaine` soit en utilisant **l'indexation positive** `chaine[0]` soit en utilisant **l'indexation négative** `chaine[-6]`.

Sous chaîne

On peut **extraire une partie de la chaîne** en utilisant l'opérateur `[:]`.

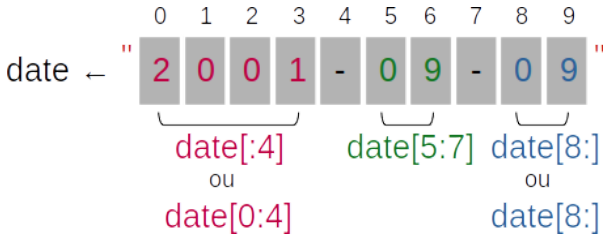


Figure 5, Extraction d'une sous-chaîne

Pour extraire une partie d'une chaîne on écrit `chaine[deb:fin]`. Cette instruction extrait les caractères compris entre les indices **deb** inclu et **fin** non inclu, **deb** est l'indice de début, et **fin** est l'indice de fin.

Recherche de position

Pour retrouver la position d'une sous-chaine `sch` dans une chaine `ch` on écrit `ch.find(sch)`, cette instruction retourne la première occurence de `sch` dans `ch`, ou `-1` si elle est introuvable.

Recherche d'existence

Pour tester l'existance d'une sous-chaîne `sch` dans une autre chaine `ch`, on écrit `sch in ch`. Cette expression renvoie un booléen : `True` / `False`.

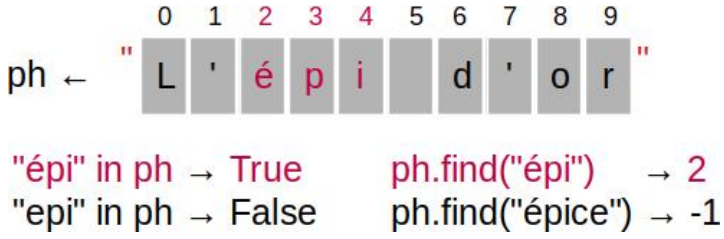


Figure 6, Recherche dans une chaîne

Effacer une sous-chaîne

Il n'existe pas une instruction Python pour **effacer une partie d'une chaîne**. Cependant, il est possible d'utiliser une concaténation de sous-chaines.
Pour effacer la partie comprise entre les indices `deb` (inclu) et `fin` (non inclu) dans une `chaine`, on écrit `chaine[:deb] + chaine[fin:]`.

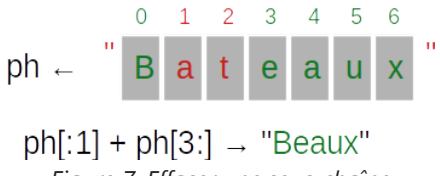


Figure 7, Effacer une sous-chaîne

Changer la casse

Python offre plusieurs méthodes pour changer la casse d'une chaine de caractère, voici quelques-unes.

- `chaine.upper()` : CONVERTIT TOUTE UNE CHAINE EN MAJUSCULES.
- `chaine.lower()` : convertit toute une chaine en minuscules.
- `chaine.capitalize()` : Met la première lettre en majuscule et les toutes autres en minuscules.
- `chaine.title()` : Convertit La Première Lettre De Chaque Mot En Majuscules.

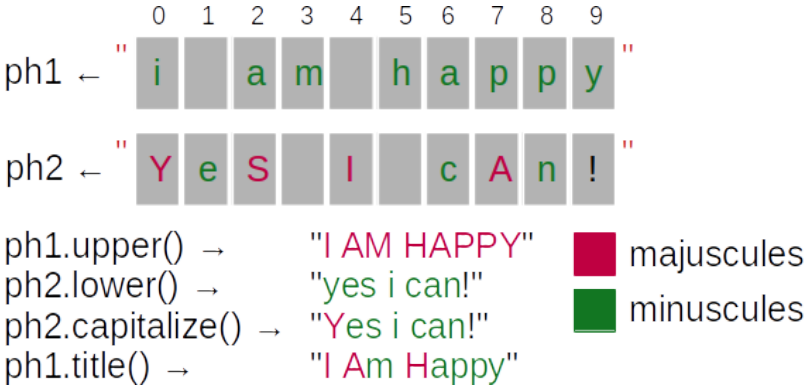


Figure 8, Modifier la casse d'une chaîne

Conversions

Une **chaîne** qui *contient une valeur numérique* peut-être transformée en **un entier** à l'aide de la fonction `int(chaine)` ou en **un réel** à l'aide de la fonction `float(chaine)`. La fonction `str(nombre)` est utilisée pour convertir un **entier** ou un **réel** en une **chaîne de caractères**.

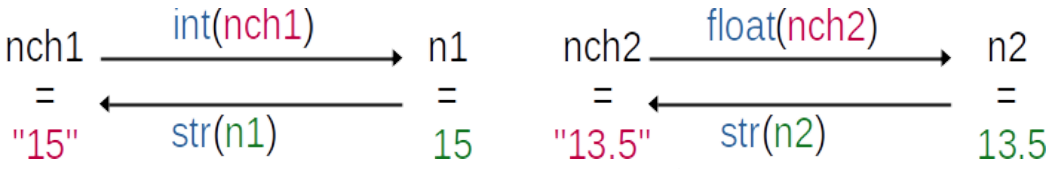


Figure 9, Conversions d'une chaîne

La conversion d'une chaîne qui ne contient pas une valeur numérique produit **une erreur d'exécution** qui peut interrompre l'exécution d'un programme pour cela on peut utiliser la fonction `chaine.isdigit()` pour tester si une chaine **contient uniquement des chiffres**.

Type entier

Le type entier couvre une partie de l'ensemble des entiers relatifs $\mathbb{Z}$, c'est à dire l'ensembles des entiers positifs 0, 1, 2, 3, etc. et des entiers négatifs -1, -2, -3, etc.
Les opérateurs possibles avec les entiers sont :

- **L'addition** `8 + 5 → 13`
- **La soustraction** `8 - 5 → 3`
- **La multiplication** `8 * 5 → 40`
- **La division réelle** `8 / 5 → 1.6`
- **Le quotient de la division entière** `8 // 5 → 1`
- **Le reste de la divion entière** `18 % 7 → 4`
- **L'exponentiation** `5 ** 3 → 125`

Voici quelques fonctions possibles avec les entiers :

- **La valeur absolue** `abs(-9) → 9`
- **Choix d'une valeur aléatoire** dans un intervalle `randint(1, 6) → une valeur au hasard dans l'intervalle [1, 6]`.
  Attention
  Pour utiliser `randint(a, b)`, il faut veiller à importer cette fonction du module `random` via l'instruction `from random import randint`.

Type réel

Le type réel couvre une partie de l'ensemble des réels $\mathbb{R}$, c'est à dire les nombres à virgule flottante.
Les opérateurs possibles avec les réels sont **les mêmes que ceux disponibles avec les entiers**, voir opérateurs avec les entiers.
Certaines fonctions avec les réels doivent être importées depuis le module `math`. Parmi ces fonctions on cite :

- **La racine carré** `sqrt(5) → 2.23606797749979`
- **La partie entière** d'un réel `trunc(6.2545) → 6`

Type booléen

Les variables de ce type peuvent prendre uniquement les deux valeurs `True` ou `False`.
On utilise les **opérateurs de comparaison** pour comparer des valeurs de mêmes types ou de types compatibles :

Opérateur	Description
<	Inférieur
<=	Inférieur ou égal
>	Supérieur
>=	Supérieur ou égal
==	égal
!=	Différent

On utilise les **opérateurs logiques** pour construire des expressions booléennes plus sophistiquées.

Opérateur	Description
not a	Négation logique, si `a` est vraie `not a` est fausse, sinon `not a` est vraie.
a and b	Et logique, `a and b` est vraie uniquement lorsque `a` est vraie et `b` est vraie.
a or b	Ou logique, `a or b` est fausse uniquement lorsque `a` est fausse et `b` est fausse, elle est vraie lorsque l'un des **opérandes** `a` ou `b` est vrai.

Tables de vérité

Opérateur unaire

a	not a
False	True
True	False

Opérateurs binaires

a	b	a and b	a or b
False	False	False	False
True	False	False	True
False	True	False	True
True	True	True	True

Priorité des opérateurs

Les opérateurs logiques sont ordonnés dans le tableau précédent selon leurs **priorités** respectives.
Il est **vivement conseillé** d'utiliser les parenthèses pour imposer l'ordre d'évaluation des opérateurs.
Exemple, l'expression `not a or b and c` est évaluée comme suit `(not a) or (b and c)`. Si on veut écrire `not ((a or b) and c)` ou `(not (a or b)) and c` , il faudra utiliser impérativement les parenthèses.

a	b	c	not a or b and c	(not a) or (b and c)	not ((a or b) and c)	(not (a or b)) and c
False	False	False	True	True	True	False
False	False	True	True	True	True	True
False	True	False	True	True	True	False
False	True	True	True	True	False	False
True	False	False	False	False	True	False
True	False	True	False	False	False	False
True	True	False	False	False	True	False
True	True	True	True	True	False	False

Comparaison des variables de type de numérique

Pour les entiers et les réels `a > b` si et seulement si `a - b > 0`, `a == b` si et seulement si `a - b == 0`.

Comparaison des caractères et des chaînes de caractères

La comparaison pour les **caractères** est basée sur le **code ASCII/Unicode**.

```
" " < "!" < ... < "0" < "1" < ... < "A" < "B" < ... < "a" < "b" < ...
```

Pour les **chaînes de caractères** on applique le même principe avec chacune des lettres des deux chaînes.
`"Amine" < "Eya"` puisque `"A" < "E"`
`"bassine" < "bassins"` bien qu'il soient de même longueur mais la dernière lettre `"e" < "s"`

Renforcement