VALEURS ET TYPES — LIST, TUPLE, DICT (VOIR PAGES SUIVANTES)

Types numériques

int	a = 5	integer (entier compris entre -∞ … +∞)
float	c = 5.6 , c = 4.3e2	floating point number (nombre décimal)
complex	d = 5 + 4j	complex numbers (nombres complexes)

Strings (Types d'objets itérables, mais non modifiables)

str	e = "hello"	Character string, chaîne de caractères

Conversion de type

int(s)	convertir chaîne s en nombre entier
float(s)	convertir chaîne s en nombre décimal
str(number)	convertir nombre entier/décimal en string
list(x)	convert tuple, range or similar to list

Noms des variables ⇨ case sensitive (différence entre caractères majuscules et minuscules)

Certains mots réservés ne sont pas autorisés :

False, None, True, and, as, assert, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while

(**print**, **sum** ⇨ not recommended, else internal functions will be overridden)

lettres (a…z , A…Z) chiffres (0…9) _ (underscore, blanc souligné)	caractères autorisés, doit commencer par une lettre
i, x	boucles et indices ⇨ lettres seules en minuscule
get_index()	modules, variables, fonctions et méthodes ⇨ minuscules + blanc souligné
MAX_SIZE	(pseudo) constantes ⇨ majuscules et blanc souligné
CamelCase	nom des classes ⇨ CamelCase

CHAÎNES DE CARACTÈRES (=SÉQUENCES NON-MODIFIABLES, IMMUTABLE)

Les caractères d'une chaîne ne peuvent pas être modifiés. Python ne connaît pas de caractères. Un caractère isolé = chaîne de longueur 1. Dans les exemples suivants : s = chaîne de caractères

- ord('A') ⇨ return integer Unicode code point for char (e.g. 65)
- chr(65) ⇨ return string representing char at that point (e.g. 'A')

String literals

"texte" ou 'texte'	délimiteurs doivent être identiques
""" chaîne sur plusieurs lignes """	chaîne sur plusieurs lignes, délimitée par """ ou '''
"abc\"def" ou 'abc\'def'	inclure le délimiteur dans la chaîne
\n	passage à la ligne suivante
\\	pour afficher un \

Caractères et sous-chaînes (Voir les exemples sous ⇨ Listes-Affichage)

Opérateurs

"abc" + "def" ou "abc" "def"	⇨ "abcdef" (concaténation)
"abc" * 3 ou 3 * "abc"	⇨ "abcabcabc" (multiplication)

Affichage ⇨ f-string (formatted strings), chaîne de char. préfixée par f ou F

f"{var1} x {var2} = {var1 * var2}"	{…} = remplacé par variables ou expression
"{} x {} = {}".format(v1, v2, v1*v2)	ou bien : str.format()
"{0}{1}{0}".format('abra', 'cad')	⇨ "abracadabra" (on peut aussi les numéroter)

Placeholder options

{:format-spec}	{:4} ou {:>4} ⇨ padding of 4, right aligned
format-spec is : [fill]align	{:.5} ⇨ truncate to 5 chars
fill = espace (par défaut)	{:10.5} ⇨ padding of 10, truncate to 5
	{:.2f} ⇨ display as float with 2 decimals
{x:3d} ⇨ display as integer, padding = 3	{:6.2f} ⇨ float with 2 decimals, padding = 6

align:	< left-aligned	=	padding after sign, but before numbers
	> right-aligned (default for numbers)	^	centered

Utiliser une variable var1 dans format-spec : "…{:{var1}}…".format(…, var1 = value, …)

Méthodes

s.capitalize()	renvoie une copie avec le premier caractère en majuscule
s.lower() | s.upper()	renvoie une copie en lettres minuscules | majuscules
s.strip()	renvoie une copie et enlève les caractères invisibles (whitespace) au début et à la fin de s
s.strip(chars)	renvoie une copie et enlève les caractères chars au début et à la fin de s
s.split()	renvoie une liste des mots (délimités par whitespace), pas de mots vides
s.split(sep)	renvoie une liste des mots (délimités par sep), sous-chaînes vides si plusieurs sep consécutifs
s.find(sub[, start[, end]])	renvoie l'indice de la 1ère occurrence de sub dans la sous-chaîne [start:end] de s, renvoie -1 si pas trouvé
s.index(sub[, start[, end]])	idem, mais exception ValueError si pas trouvé
s.replace(old, new[, n])	renvoie une copie avec les n (default = toutes) premières occurrences de old remplacés par new
s.isalpha()	True si au moins un caractère et que des lettres
s.isdigit()	True si au moins un chiffre et que des chiffres
s.isalnum()	True si au moins un caractère et que des lettres ou chiffres
s.islower()	True si au moins une lettre et que des minuscules
s.isupper()	True si au moins une lettre et que des majuscules
s.isspace()	True si au moins un whitespace et que des whitespace
for char in s :	parcourir les lettres de la chaîne de caractères
s.join(iterable)	returns a string created by joining the elements of an iterable by string separator
"xx".join("123") ⇨ "1xx2xx3"	
s.join([str(elem) for elem in lst])	convertir liste en chaîne avec séparateur s

LISTES (=SÉQUENCES MODIFIABLES) ⇨ [] — type: list

Dans une même liste ⇨ variables de différents types = possible.

Création — * = unpack operator

lst = []	créer une liste vide
lst = [item1, item2, …] , lst = [23, 45]	créer une liste avec des éléments
new_lst = lst1 + lst2 (= [*lst1, *lst2])	Attention : **crée une nouvelle liste**
list(x), ex : lst = list(range(5))	Convertir uplet, range ou semblable en liste

Remarque

A = B = [] ⇨ A = [] et B = A	les 2 noms (A et B) pointent vers la même liste

list comprehensions (computed lists)

lst = [expr for var in sequence] lst = [expr for var in sequence if …]	expr is evaluated once for every item in sequence, (if is optional)

Exemple : création d'une matrice 3x3

p = [x[:] for x in [[0]*3]*3] ou p = [[0,0,0], [0,0,0], [0,0,0]]	1. construire 3 vecteurs, chacun avec 3 composants nuls 2. une copie est placée dans p, pour obtenir 3 vecteurs-lignes indépendants, ne pointant pas sur le même objet

Affichage et sous-listes — premier élément d'une liste ⇨ index 0

lst[index]	retourne l'élément à la position index (un index < 0 ⇨ accède aux éléments à partir de la fin)
lst[start :end]	retourne une sous-liste de l'indice start à end (non compris)
lst[start :end :step]	(seuls les éléments avec index = multiple de step inclus)

lst[-1]	retourne le dernier élément de lst
lst[2:-1]	sous-liste à partir de l'indice 2 jusqu'à l'avant dernier
lst[:4]	sous-liste à partir du début jusqu'à l'indice 3
lst[4:]	sous-liste à partir de l'indice 4 jusqu'à la fin
lst[:]	retourne la liste entière, pour copier une liste dans une autre variable
lst[::2]	retourne sous-liste des éléments à index pair
lst[::-1]	retourne sous-liste des éléments dans l'ordre inverse

Pour copier une liste

lst = [2, 3, 4, 5]	1st level copy (copie = lst ne fonctionne pas, car
copie = lst[:] ou copie = lst.copy()	variables pointent alors sur la même liste)
copie = [x[:] for x in lst]	copier une liste de listes (2nd level copy, shallow copy)
copie = copy.deepcopy(lst)	import copy (any level copy, deep copy)

Modification

lst[index] = item	modifie l'élément à la position index
lst[start :end] = […]	remplace la sous-liste à partir de start jusqu'à end (exclu), même de taille différente
lst.append(item)	add item as single element to end of **existing** list
lst.extend(iterable)	add each element of iterable (all items) to the existing
lst += [item1, …, item_n]	list by iterating over the argument
lst = lst + [item1, …, item_n]	Attention: create new list and add all items from both
del lst[index] , del(lst[index])	supprime l'élément à la position index
lst.remove(item)	supprime le premier élément avec la valeur item
lst.pop()	enlève et retourne le dernier élément de la liste (à la
lst.pop(index)	position indiquée par index)
lst.reverse()	inverse les items d'une liste (modifie la liste)
new_lst = reversed(lst)	retourne une liste inversée (lst = unchanged)
lst.sort()	trier la liste (modifie la liste)
new_lst =sorted(lst)	retourne une liste triée (lst = unchanged)
lst.insert(index, item)	insère l'item à la position donnée par index

Attention :

lst = [1, 2, 3, 4] lst[2] = [7,8,9] ⇨ [1, 2, [7, 8, 9], 4] (liste imbriquée)	lst = [1, 2, 3, 4] lst[2:2] = [7,8,9] ⇨ [1, 2, 7, 8, 9 , 4] (élément remplacé par plusieurs éléments)

Divers

print(lst)	affiche le contenu de la liste
len(lst)	nombre d'items dans lst
lst.count(item)	nombre d'occurrences de la valeur item
lst.index(item)	retourne l'index de la 1ère occurrence de item, sinon ⇨ exception ValueError
item in lst (item not in lst)	indique si l'item se trouve dans lst (n'est pas dans)
min(lst) / max(lst)	retourne l'élément avec la valeur min. / max.
sum(lst[,start])	retourne la somme à partir de start (= 0 par défaut)
for item in lst:	parcourir les éléments
for index in range(len(lst)):	parcourir les indices
for index, item in enumerate(lst):	parcourir l'indice et les éléments
for item in reversed(lst):	parcourir dans l'ordre inverse
for item in lst[:]:	effacer éléments d'une liste ⇨ utiliser copie de lst
for i in range(len(lst)-1, -1, -1): … code pour effacer des items	effacer certains éléments d'une liste ⇨ il faut parcourir la liste de la fin au début, si on a besoin de l'index

while i < len(lst):
if … code pour effacer items
else:
i = i + 1
	effacer certains éléments d'une liste

if lst: ou if len(lst) > 0:	test si la liste lst n'est pas vide

RANGE (=SÉQUENCES NON MODIFIABLES)

Retourne une séquence non modifiable d'entiers

range([start], stop[, step]) (start, stop, step = integers)	retourne une séquence d'entiers sans la valeur stop range(n)⇨ [0,1,2, …, n-1], ex.: range(3) ⇨ [0, 1, 2] range(2, 5) ⇨ [2, 3, 4] range(0, -10, -2) ⇨ [0, -2, 4, -6, -8]

LES UPLETS (TUPLES) -> () type: tuple

Uplet = collection d'éléments séparés par des virgules. Comme les chaînes **pas modifiables**

Création

`tuple = (a, b, b, …)` `t = ("a", 2.4, 45)`	créer un uplet
`tuple = a, b, c, …` `t = (1,) ou t = 1,`	(on peut omettre les parenthèses, si clair)
`tuple1 = tuple2`	copier un uplet

Extraction

`(x, y, z) = tuple` ou `x, y, z = tuple`	extraire les éléments d'un uplet

Affichage ⇨ voir listes

Premier élément d'un uplet ⇨ index 0

`tuple[index]`	retourne l'élément à la position index (un index < 0 ⇨ accède aux éléments à partir de la fin)
`tuple[start:end]`	retourne un sous-uplet de l'indice [start ; end[

LES DICTIONNAIRES -> {} type: dict

Les dictionnaires sont modifiables, mais pas des séquences. L'ordre des éléments est aléatoire. Pour accéder aux objets contenus dans le dictionnaire on utilise des clés (keys). Classe : `dict`

Création

`dic = {}` ou `dic = dic()`	créer un dictionnaire vide
`dic = {key1: val1, key2: val2, …}`	créer un dictionnaire déjà rempli : `d = {"nom":"John", "age":24}`
`dic[key] = value`	ajouter une clé:valeur au dictionnaire si la clé n'existe pas encore, sinon elle est remplacée

key peut être alphabétique, numérique ou type composé (ex. uplet)

Affichage

`dic[key]`	retourne la valeur de la clé keys. Si la clé n'existe pas une exception `KeyError` est levée
`dic.get(key, default = None)`	retourne la valeur de la clé, sinon `None` (ou la valeur spécifiée comme 2e paramètre de get)
`dic.keys()`	retourne les clés du dictionnaire
`list(dic.keys())` ou `list(dic)`	… comme liste
`tuple(dic.keys())`	… comme uplet
`sorted(dic.keys())` `sorted(dic)`	renvoie une liste des clés dans l'ordre lexicographique
`dic.values()` `list(dic.values())`	renvoie les valeurs du dictionnaire / comme liste
`dic.items()` `list(dic.items())`	renvoie les éléments du dictionnaire sous forme d'une séquence de couples / d'une liste de couples

Modification

`dic[key] = value`	ajouter une clé:valeur au dictionnaire, si la clé n'existe pas encore (sinon elle est remplacée)
`del dic[key]` ou `del(dic[key])`	supprime la clé key du dictionnaire
`dic.pop(key)`	supprime la clé key du dictionnaire et renvoie la valeur supprimée

Divers

`len(dic)`	renvoie le nombre d'éléments dans le dictionnaire
`if key in dic: , if key not in dic`	tester si le dictionnaire contient une certaine clé
`for c in dic.keys():` ou `for c in dic:`	parcourir les clés d'un dictionnaire
`for c, v in dic.items():`	parcourir les éléments du dictionnaire
`copie = dic.copy()`	crée une copie (shallow copy) du dictionnaire (une affectation crée seulement un nouveau pointeur sur le même dictionnaire) - 1st level copy
`copie = copy.deepcopy(dic)`	`import copy` (any level copy)
`max(dic, key=len)`	retourne la clé la plus longue
`dic1.update(dic2)`	combine 2 dictionnaires en un seul (dic1), les clés de dic2 sont prioritaires

Expressions et opérateurs

Opérateurs entourés d'espaces. Utiliser des parenthèses pour grouper des opérations (modifier la priorité)

Opérateurs mathématiques

La 1ère colonne indique la priorité des opérateurs

1.	`**`	exponentiation	`6 ** 4` ⇨ `1296`
2.	`-, +`	signe	`-5`
3.	`*`	multiplication	`x *= 3` ⇨ `x = x * 3`
	`/`	division (entière ou réelle)	`x /= 3` ⇨ `x = x / 3`
	`//`	quotient de la division entière (arrondit vers le négatif infini)	`6 // 4` ⇨ `1` `-6.5 // 4.1` ⇨ `-2.0`
	`%`	modulo, reste (positif) de la division entière obtient le signe du diviseur	`6 % 4` ⇨ `2`, `-6.5 % 4. 1` ⇨ `1.7` `6 % -4` ⇨ `-2`
4.	`+`	addition	`x += 3` ⇨ `x = x + 3`
	`-`	soustraction	`x -= 3` ⇨ `x = x - 3`

Opérateurs relationnels

retournent `True` ou `1` si l'expression est vérifiée, sinon `False` ou `0`

5.	`==`	égal à	
	`!=`	différent de	
	`>`	strictement supérieur à	
	`<`	strictement inférieur à	
	`>=`	supérieur ou égal à	(exemple : `x >= a` ou `b >= x >= a` pour `a <= b`)
	`<=`	inférieur ou égal à	(exemple : `x <= b` ou `a <= x <= b`)

chaînes de caractères ⇨ ordre lexicographique, majuscules précèdent les minuscules

Opérateurs logiques

6.	`not x`	**non** (retourne `True`, si x est faux, sinon `False`)
7.	`x and y`	**et** (retourne x, si x est faux, sinon y) `and` ne vérifie le 2e argument que si le 1er argument est vrai
8.	`x or y`	**ou** (retourne y, si x est faux, sinon x) `or` ne vérifie le 2e argument que si le 1er argument est faux

Affectation

L'affectation attribue un type bien déterminé à une variable.

`variable = expression`	Affectation simple, attribuer une valeur à une variable
`a = b = c = 1`	affectation multiple
`x, y = 12, 14`	affectation parallèle
`x, y = y, x`	échanger les valeurs des 2 variables (swap)

Entrée / Sortie

Entrée

`var = input()`	renvoie une chaîne de caractères
`var = input(message)`	renvoie une chaîne de caractères et affiche le message
`int = int(input(…))`	renvoie un entier
`float = float(input(…))`	renvoie un nombre décimal

Sortie

`print(text, end="final")`	affiche `text` et termine avec `final` (par défaut `end="\n"`)
`print("abc", "def")`	⇨ `abc def` (arguments séparés par espace, nouvelle ligne)
`print("abc", end="+")`	⇨ `abc+` (pas de passage à la ligne)
`print(var)`	`var` est converti en chaîne et affichée
`print("value=", var)`	affiche le texte suivi d'une espace, puis de la valeur de `var`
`print()`	simple passage à la ligne
`print(str * n) | print(n * str)`	afficher n fois le texte `str`

Les commentaires

`# commentaire`	sur une seule ligne
`'''comments'''` ou `"""comments"""`	sur plusieures lignes (= string literal)

Structure alternative et répétitive

Structure alternative

`if condition1:`
`  instruction(s)`
`elif condition2:`
`  instructions(s)`
`…`
`else:`
`  instruction(s)`
	• exécute seulement les instructions, où la condition est vérifiée
	• si aucune condition n'est vérifiée, les instructions de `else` sont exécutées
	• `else` et `elif` sont optionnels

`<on true> if <expr> else <on false>`	• ternary operator (opérateur ternaire)

Structure répétitive (boucle for)

`for itérateur in liste de valeurs:`
`  instruction(s)`
`for i in range (10):  # values 0, 1, … 9`
`for _ in range (10):  # values 0, 1, … 9`
	• répète les instructions pour chaque élément de la liste
	• nombre de répétitions = connu au départ
	• _ si valeur de l'itérateur n'est pas utilisée

Structure répétitive (boucle while)

`while condition(s):` `  instruction(s)`	• répète les instructions tant que la condition est vraie • **pour pouvoir sortir de la boucle, la variable utilisée dans la condition doit changer de valeur** • nombre de répétitions != connu au départ

A l'intérieur d'une boucle **for** ou **while**

`break`	quitte la boucle immédiatement
`continue`	continue avec la prochaine itération

Les fonctions

Le code de la fonction doit être placé plus haut dans le code source (avant l'appel de la fonction).

- arguments simples (nombres, chaînes, uplets) ⇨ passage par valeur (valeurs copiés)
- arguments complexes (listes, dictionnaires) ⇨ passage par référence (vers les originaux)

Définition et appels

`def my_function(par1, …, par_n):`
`  instruction(s)`
`  …`
`  return var`
	définit une fonction `my_function`
	• par1 … par_n sont les paramètres
	• une ou plusieurs instructions `return`…
	• peut renvoyer plusieurs réponses (uplet, liste)
	Si la fonction ne contient pas d'instruction `return`, la valeur `None` est renvoyée

`my_function(arg1, … arg_n)` `var = my_function(arg1, … arg_n)`	appel de la fonction, arguments affectés aux paramètres dans le même ordre d'apparition
`my_function(*lst)`	* to unpack list elements
`my_function(**dct)`	* to unpack dictionary elements

`def func(par1, …, par_n = val):`
`ex : def add(elem, to = None):`
`  if to is None:`
`  to = []`
	paramètre par défaut
	ATTENTION:
	`  def add(elem, to = []):`
	ne marche pas, car les arguments par défaut ne sont évalués qu'une seule fois et réutilisés pour tous les appels de la fonction.

`def func(par1, …, *par_n):`	*par_n = nombre variable de paramètres (liste)

* = unpack operator to unpack list elements https://docs.python-guide.org/writing/gotchas/

Variables globales

Les paramètres et variables locales cachent les variables globales/extérieures.

`def func(…):` `  global var`	var est déclaré comme variable global, la variable `var` à l'extérieur de la boucle est donc modifiée/utilisée

UTILISATION DE MODULES (BIBLIOTHÈQUES)

Utiliser des modules

`import module`	importe tout le module, il faut préfixer par le nom du module . Ex : `import math` ⇨ `math.sqrt()`
`import module as name`	
`from module import *` `*** à éviter ***`	intègre toutes les méthodes de module, pas besoin de préfixer le nom du module ex : `from math import *` ⇨ `sqrt()`
`from module import m1, m2, …` `from math import sqrt, cos`	intègre seulement les méthodes mentionnées ⇨ `sqrt(…)`, `cos(…)`

MODULE: MATH — import math

Built-in functions (no import required)

`abs(x)`	valeur absolue (aussi nombres complexes)
`round(x)`	x est arrondie vers l'entier pair le plus proche • `round(3.5)` ⇨ 4 (rounds to nearest EVEN integer) • `round(4.5)` ⇨ 4 (rounds to nearest EVEN integer)

import math

`math.pi`	le nombre pi
`math.cos(x)` / `.sin(x)` / `.tan(x)`	cosinus/sinus/tangente d'un angle en radian
`math.sqrt(x)`	racine carrée
`math.fabs(x)`	valeur absolue ⇨ retourne un float
`math.ceil(x)` / `math.floor(x)`	x est arrondie vers le haut / vers le bas
`math.trunc(x)`	retourne l'entier sans partie décimale
`math.pow(x, y)`	x exposant y
`math.gcd(x, y)`	retourne le PGCD des 2 nombres

MODULE: RANDOM — import random

`random.randint(a, b)`	retourne un entier au hasard dans l'intervalle [a ; b]
`random.random()`	retourne un réel au hasard dans l'intervalle [0 ; 1[
`random.uniform(a, b)`	retourne un réel au hasard dans l'intervalle [a ; b]
`random.choice(seq)`	retourne un élément au hasard de la séquence seq (si seq est vide ⇨ exception `IndexError`)
`random.sample(seq, k)`	retourne une liste de k éléments uniques (choisis au hasard) de la séquence seq
`random.randrange(stop)`	retourne un entier au hasard de [`start ; stop`[. Seuls
`random.randrange(start, stop)`	les multiples de `step` sont possibles.
`radnom.randrange(start, stop, step)`	(`start` = 0, `step` = 1 par défaut)
`random.shuffle(seq)`	mélange aléatoirement les éléments de seq

retourner le nombre aléatoire −1 ou 1

`randrange(-1, 2, 2)`	`2 * randint(0, 1) - 1`	`h = [-1, 1] , choice(h)`

MODULE: TIMIT — import timit

```
t1_start = timeit.default_timer()
  …
t2_stop = timeit.default_timer()
print(t2_stop - t1_start)
```

Return process time of current process as float in seconds

LES FICHIERS

Entrées/sorites console et redirection

`STDIN`	entrée standard ⇨ le clavier (pour entrer des données)
`STDOUT`	sortie standard ⇨ l'écran (pour afficher les résultats)
`STDERR`	l'écran (pour envoyer les messages d'erreur)
`command > filename`	rediriger la sortie standard vers un fichier (créé/remplacé)
`command >> filename`	rediriger la sortie standard vers un fichier (ajouté)
`command > NUL`	annuler sortie vers STDOUT
`command < filename`	rediriger entrée depuis un fichier

Tubes et filtres

`command1 | command2`	rediriger la sortie de command1 comme entrée à command2

Manipulation de fichiers

`file = open(filename, mode='r')`	retourne un objet fichier, `'r'` = mode lecture, `'w'` = mode écriture, `'a'` = mode écriture/ajout (à la fin)
`line = file.readline()`	lit et retourne la prochaine ligne complète avec caractère fin de ligne (retourne une chaîne vide `""` si la fin du fichier est atteinte)
`for line in file:` `  …`	lit tout le fichier ligne après ligne (voir ci-dessous)
`line = file.readline()` `while line != "":` `  …` `  line = file.readline()`	lit tout le fichier ligne après ligne ⇨ utiliser `line.strip()` pour enlever les caractères invisibles (espaces, newline) au début et à la fin d'une ligne
`lines_list = file.readlines()`	lit tout le fichier et retourne une liste de chaînes
`file.read()`	lit tout le fichier et retourne une chaîne
`file.write(str)`	écrit dans `file` la chaîne `str`
`file.close()`	fermer `file` (si traitement du fichier est terminé)

Lire de STDIN en Python (manière de filtres)

```
import sys
line = sys.stdin.readline()
while line != "":
  …
  line = sys.stdin.readline()
```

```
import sys
for line in sys.stdin:
  …
```

lire les données de STDIN, ou

To terminate readline(), when STDIN is read from keyboard, press CTRL-D (CTRL-Z on Windows)

MODULE: STRING — import string

`string.ascii_uppercase`	chaîne de caractères pré-initialisée avec `'ABCDEF … XYZ'`
`string.ascii_lowercase`	chaîne de caractères pré-initialisée avec `'abcdef … xyz'`

MODULE: SYS — import sys

`sys.stdin.readline()`	lit la prochaine ligne de STDIN (`''` si EOF)
`sys.maxsize`	valeur max. d'un entier en Python (32-bit ⇨ 2^{31}, 64-bit ⇨ 2^{63})
`sys.setrecursionlimit(limit)`	définir la profondeur maximale de la pile lors d'appels récursifs

MODULE: COPY — import copy

`copie = copy.deepcopy(x)`	renvoie une copie récursive (ou profonde) de x (= copie de l'objet et copies des objet trouvés dans l'objet original)

MODULES ET LIBRAIRIES (PACKAGES)

Modules

⇨ fichiers dans lesquels on regroupe différentes fonctions

1. créer un fichier (module) contenant des fonctions 2. dans un 2e fichier utiliser : import module	⇨ utiliser les fonctions du module **Attention** : lors de modifications dans le module, il faut d'abord supprimer le fichier avec l'extension `.pyc` dans le dossier : `__pycache__`

Librairies (packages)

⇨ dossier complet pour gérer les modules, peuvent contenir d'autres dossiers

⇨ dossier principal doit contenir le fichier vide nommé `__init__.py`

1. créer un dossier 2. ajouter des modules 3. créer le fichier vide `__init__.py` dans le dossier	⇨ créer une librairie

Installer des librairies (packages) externes

PyCharm

⇨ `File` -> `Settings` -> `Project:` votre projet actuel

⇨ Sélectionner l'interprétateur Python (p.ex. 3.6.1), puis cliquer sur le symbole + à droite

⇨ Choisir libraire à installer dans la liste (cocher "Install to user's site packages directory" si pas administrateur)

Thonny

⇨ `Tools` -> `Manage Packages…`

⇨ Entrez le nom de la librairie pour la rechercher et cliquer sur Install

PACKAGE : PILLOW — from PIL import image

Module : Image (https://pillow.readthedocs.io/en/5.1.x/)

`PIL.Image.open(fp, mode="r")`	ouvre l'image fp et retourne un objet Image
`PIL.Image.new(mode, size, color=0)`	crée un nouveau objet image et le retourne • mode : `'RGB'` ⇨ 3x8 bit pixels, true color • `size` = uplet (largeur, hauteur)
`Image.crop(box=None)`	retourne une région rectangulaire • box = uplet (left, upper, right, lower)
`Image.paste(im, box=None, mask=None)`	copie l'image `im` sur cet image • box = uplet (left, upper) ou (left, upper, right, lower)
`Image.save(fp, format=None, **params)`	enregistre l'image sous le nom `fp`

PROGRAMMATION ORIENTÉ OBJET (POO)

OOP = object oriented programming, Python = langage orienté objet hybride

Objet

Objet = structure de données valuées et cachées qui répond à un ensemble de messages

- **attributs** = données/champs qui décrivent la structure interne
- **interface de l'objet** = ensemble des messages
- **méthodes** = réponse à la réception d'un message par un objet

Principe d'encapsulation ⇨ certains attributs/méthodes sont cachés

- **Partie publique** ⇨ visible et accessible par tous
- **Partie privée** ⇨ seulement accessible et utilisable par les fonctions membres de l'objet (invisible et inaccessible en dehors de l'objet)

Principe de masquage d'information ⇨ cacher comment l'objet est implémenté, seul son interface publique est accessible.

Classe (= définition d'un objet)

Instanciation ⇨ création d'un objet à partir d'une classe existante (chaque objet occupe une place dans la mémoire de l'ordinateur)

`class ClassName:` `def __init__(self, par1, … par_n):` `  self.var1 = …` `  self.var2 = …` `def __str__(self):` `  …` `  return chaîne_de_texte` `def method1(self, …):` `  …` `  return result`	définit la classe `ClassName` (CamelCase) les fonctions sont appelées **méthodes** • `__init__()` ⇨ constructeur, appelé lors de l'instanciation • `__str__(self)` ⇨ string representation of object, e.g. `print(object)` • `self` doit être le 1er paramètre et référencie la classe elle-même • `self.var…` ⇨ attributs, accessibles de l'extérieur • `method…()` ⇨ méthodes, accessible de l'extérieur **Convention** : utiliser le préfix (`_`) si des attributs ou méthodes ne doivent pas être accédés de l'extérieur (même s'ils sont toujours accessibles)
`obj = ClassName(…)`	instancie un nouvel objet de la classe dans la mémoire
`obj.method(…)`	appel de la méthode de l'objet (self = `obj` est toujours passé comme 1er paramètre)

Récursivité

- Algorithme récursif ⇨ algorithme qui fait appel(s) à lui-même
- Attention : il faut prévoir une condition d'arrêt (= cas de base)
- Pour changer la limite max. de récursions ⇨ voir module **sys**

PYGAME — BIBLIOTHÈQUE POUR CRÉER DES JEUX

Structure d'un programme Pygame

```
import pygame, sys                          # Initialisation
from pygame.locals import *                 importer les librairies et initialiser les modules de pygame
pygame.init()
WIDTH = …                                   # Création de la surface de dessin
HEIGHT = …                                  définir la largeur (0…WIDTH-1) et la hauteur (0… HEIGHT-1) de la fenêtre et retourner un objet de type surface
size = (WIDTH, HEIGHT)
screen = pygame.display.set_mode(size)
pygame.display.set_caption(str)             # Titre de la fenêtre
                                            définir le titre de la fenêtre
screen.fill(color)                          # Effacer surface de dessin
                                            remplir arrière-plan avec couleur
FPS = frequence   # en Hz                   # Fréquence d'image
clock = pygame.time.Clock()                 créer l'objet clock avant la boucle
done = False                                # Boucle principale
while not done:                             boucle principale (infinie)
for event in pygame.event.get():            # Gestion des événements
  if event.type == QUIT:                    Event loop
    done = True                             • Gestion de tous les événements dans une seule boucle for à l'intérieur de la boucle principale.
  elif event.type == <type d'événement>:    • Toutes les instructions if doivent être regroupées dans une seule boucle for
    <instruction(s)>
  …
  … dessins …
  # mise à jour de l'écran
  pygame.display.update()
  # Fréquence d'image                       insère des pauses pour respecter FPS (appel à la fin de la boucle principale)
  clock.tick(FPS)
pygame.quit()                               # Fermer la fenêtre et quitter le programme
sys.exit()
```

Types d'événements — https://www.pygame.org/docs/ref/event.html

Événement de terminaison

`QUIT`
`if event.type == QUIT:`
`  …`
	L'utilisateur a cliqué sur la croix de fermeture de la fenêtre.
	Pour terminer correctement, utiliser :
	`pygame.quit()` et `sys.exit()`

Événements - clavier — https://www.pygame.org/docs/ref/key.html

`KEYDOWN / KEYUP` `if event.type == KEYUP:`	une touche du clavier est enfoncée / relâchée ⇨ `event.key`, `event.mod`
`if event.key == K_a:`	indique quelle touche a été enfoncée
`…`	`K_a`, `K_b`, … touche a, b,… (pareil pour le reste de l'alphabet)
	`K_0`, `K_1`, … touche 0, 1, … en haut (pareil pour les autres chiffres)
	`K_KP0`, `K_KP1`, … touche 0, 1, … sur pavé numérique (pareil …)
	`K_LALT`, `K_RALT` touche ALT (à gauche | à droite)
	`K_LSHIFT`, `K_RSHIFT` touche SHIFT (à gauche | à droite)
	`K_LCTRL`, `K_RCTRL` touche CONTROL (à gauche | à droite)
	`K_SPACE` touche espace
	`K_RETURN` touche ENTER
	`K_ESCAPE` touche d'échappement
	`K_UP`, `K_DOWN`, `K_LEFT`, `K_RIGHT` touches flèches
	`KMOD_NONE` no modifier keys pressed
	(can be used to reset pressed keys on KEYUP)

`keys = pygame.key.get_pressed()`
`if keys[K_LEFT] and not keys[K_RIGHT]:`
`  …`
	get state of all keyboard buttons
	(refresh with ⇨ `pygame.event.get()`)
	p. ex. faire une action aussi longtemps que la touche flèche ⇦ est enfoncée

Événements – souris — https://www.pygame.org/docs/ref/mouse.html

`MOUSEBUTTONDOWN` `MOUSEBUTTONUP`	un bouton de la souris a été enfoncé / relâché ⇨ `event.pos`, `event.button`
`MOUSEMOTION` `if event.type == MOUSE…`	la souris a été déplacée ⇨ `event.pos`, `event.rel`, `event.buttons`

Boutons de la souris

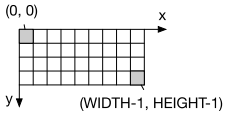
`if event.button == 1:`	indique quel bouton a déclenché l'événement 1 = left, 2 = middle, 3 = right, 4 = scroll-up, 5 = scroll-down
`pygame.mouse.get_pressed()`	retourne séquence de 3 valeurs pour l'état des 3 boutons de la souris (de gauche à droite), `True` si enfoncé. Ex. : `if pygame.mouse.get_pressed() == (True, False, False):`
`event.buttons` `if event.buttons[0]: # left b.?`	⇨ tuple for (left, middle, right) mouse buttons Ex. : (1,0,0) ⇨ value 1 if pressed, else 0

Position de la souris

`(x, y) = event.pos`	position du pointeur de souris à l'instant de l'événement
`(x, y) = pygame.mouse.get_pos()`	position actuelle du pointeur de souris (uplet)

La surface de dessin

Origine (0,0) = point supérieur gauche
- largeur de `0` … `WIDTH-1`
- hauteur de `0` … `HEIGHT-1`



Dimensions de la surface de dessin

`screen = pygame.display.get_surface()`	retourne la surface de dessin
`screen.get_width()`	retourne la largeur de la surface de dessin
`screen.get_height()`	retourne la hauteur de la surface de dessin
`w, h = screen.get_size()`	retourne les dimensions de la surface de dessin sous forme d'uplet

Couleurs

`color = Color(name)` `color = name`	renvoie la couleur du nom name (String), ex.: `"white"`, `"black"`, `"green"`, `"red"`, `"blue"`
`color = Color(red, green, blue)`	red, green, blue = nombres de 0 … 255

Obtenir la couleur d'un point (pixel)

`color = screen.get_at((x, y))`	retourne la couleur du point (pixel) à la position indiquée

Effacer/Remplir surface de dessin

`screen.fill("black")` `screen.fill(Color("black"))`	remplir arrière-plan en noir
`screen.fill("white")` `screen.fill(Color("white"))`	remplir arrière-plan en blanc

Dessiner une ligne/un point sur la surface (`screen`)

`pygame.draw.line(screen, color, start_point, end_point[, width])`
- dessiner un point si `start_point` = `end_point`
- `start_point` et `end_point` sont inclus
- `width` = 1 par défaut

`screen.set_at((x, y), color )`	dessiner un point (pixel) à la position `(x, y)`

Dessiner un rectangle sur la surface (`screen`)

`pygame.draw.rect(screen, color, rect_tuple[, width])`
- `rect_tuple` = `(x, y, width, height)` avec x, y = coin supérieur gauche
- ou `rect_tuple` = `pygame.Rect(x, y, width, height)`
- `width` = `0` par défaut (= rectangle plein)

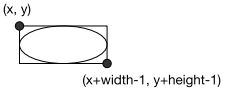
Dessiner une ellipse inscrite dans le rectangle bounding_rect sur la surface (`screen`)

`pygame.draw.ellipse(screen, color, bounding_rect[, width])`
- `bounding_rect` = `(x, y, width, height)` avec x, y = coin supérieur gauche
- ou `rect_tuple` = `pygame.Rect(x, y, width, height)`
- `width` = `0` par défaut (= ellipse pleine)

Dessiner un cercle sur la surface (`screen`)

`pygame.draw.circle(screen, color, center_point, radius[, width])`
- `center_point` = centre du cercle
- `radius` = rayon
- `width` = `0` par défaut (= cercle plein)

Remarque : `rect_tuple` et `bounding_rect`
- coordonnées du point supérieur gauche : (x, y)
- coordonnées du point inférieur droit : (x+width-1, y+height-1)



Mise à jour de la surface de dessin

`pygame.display.update()` `pygame.display.flip()`	rafraîchir la surface de dessin pour afficher les dessins
`pygame.display.update(rect)`	rafraîchir que la partie `rect` = pygame.Rect(x, y, width, height)

Gestion du temps (fréquence de rafraîchissement)

avant la boucle principale

`FPS = frequence`	définir fréquence de rafraîchissement en Hz
`clock = pygame.time.Clock()`	créer un objet de type Clock

à la fin de la boucle principale (après la mise à jour de la surface de dessin)

`clock.tick(FPS)`	insérer des pauses pour respecter la fréquence voulue

pygame.Rect

`rect = Rect(left, top, width, height)` `rect = Rect((left, top), (width, height))`	créer un nouveau objet Rect, avec `left`, `top` = coin supérieur gauche
`rect.normalize()`	corrige les dimensions négatives, le rectangle reste en place avec les coordonnées modifiées
`rect.move_ip(x, y)`	déplace `rect` de x, y pixels (retourne `None`)
`rect.move(x, y)`	retourne un nouveau `rect` déplacé de x, y pixels
`rect.contains(rect2)`	retourne `True` si `rect2` est complètement à l'intérieur de `rect`
`rect.collidepoint(x, y)` `rect.collidepoint((x, y))`	retourne `True` si le point donné se trouve à l'intérieur de `rect`
`rect.colliderect(rect2)`	retourne `True` si les 2 rectangles se touchent

Affichage de textes

`pygame.font.SysFont(name, size[, bold, italic])`	crée un objet de type Font à partir des polices système (`bold` et `italic` = `False` par défaut)
`surface = font.render(text, antialias, color[, background])`	dessine le texte `text` sur une nouvelle surface de dessin et retourne la surface (`background` = `None` par défaut)
`screen.blit(source, dest[, area, special_flags])`	copie la surface `source` sur la surface `screen` à la position `dest` (coin sup. gauche)
`pygame.display.update()`	met à jour la surface de dessin

Exemple:

`font = pygame.font.SysFont("comicsansms", 20)`	crée un objet font
`surf_text = font.render("Hello", True, "green")`	crée nouvelle surface avec texte
`screen.blit(surf_text, (100, 50))`	copie la surface `surf_text` sur `screen` à la position indiquée et mise à jour
`pygame.display.update()`	

(`surf_text.get_height()`, `surf_text.get_width()` ⇨ retourne la largeur/hauteur du texte)

Divers

`pygame.time.delay(delay)`	interrompre le programme pour un nombre de ms donnés (`delay`) et renvoie le nombre effectif de ms utilisés
`pygame.time.ticks()`	renvoie le temps en ms depuis l'appel de `pygame.init()`

ASCII CODES https://theasciicode.com.ar/

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

- `ord('A')` ⇨ return integer Unicode code point for char (e.g. 65)
- `chr(65)` ⇨ return string representing char at that point (e.g. 'A')

STRING CONSTANTS (MODULE : STRING) import string

https://docs.python.org/3/library/string.html

`string.ascii_lowercase`	all lowercase letters : `'abcdefghijklmnopqrstuvwxyz'`
`string.ascii_uppercase`	all uppercase letters : `'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`
`string.ascii_letters`	concatenation of the ascii_lowercase and ascii_uppercase constants
`string.digits`	the string `'0123456789'`
`string.hexdigits`	the string `'0123456789abcdefABCDEF'`
`string.octdigits`	the string `'01234567'`
`string.punctuation`	string of ASCII punctuation chars : `!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~.`
`string.whitespace`	string containing all ASCII whitespace (space, tab, linefeed, return, formfeed, and vertical tab)
`string.printable`	string of printable ASCII characters (combination of digits, ascii_letters, punctuation, and whitespace)

PYTHON SETS ⇨ { }

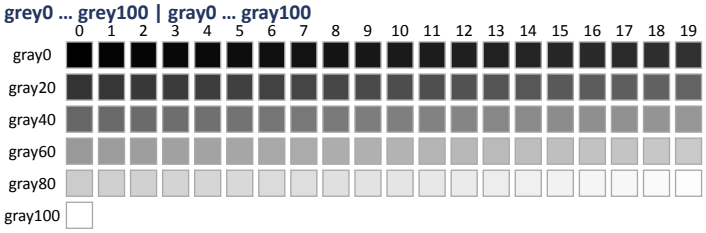
Set items are unordered, unchangeable and do not allow duplicate values. Items can be added or deleted.

`s = set()`	create empty set
`s = {"ap", "ban", "ch"}`	create set with items

PYGAME COLORS

https://github.com/pygame/pygame/blob/main/src_py/colordict.py

grey0 ... grey100 | gray0 ... gray100



color, color1, color2, color3, color4

	1	2	3	4
aliceblue				
antiquewhite				
aqua				
aquamarine				
azure				
beige				
bisque				
black				
blanchedalmond				
blue				
blueviolet				
brown				
burlywood				
cadetblue				
chartreuse				
chocolate				
coral				
cornflowerblue				
cornsilk				
crimson				
cyan				
darkblue				
darkcyan				
darkgoldenrod				
darkgray				
darkgreen				
darkgrey				
darkkhaki				
darkmagenta				
darkolivegreen				
darkorange				
darkorchid				
darkred				
darksalmon				
darkseagreen				
darkslateblue				
darkslategray				
darkslategrey				
darkturquoise				
darkviolet				
deeppink				
deepskyblue				
dimgray				
dimgrey				
dodgerblue				
firebrick				
floralwhite				
forestgreen				
fuchsia				
gainsboro				
ghostwhite				

	1	2	3	4
gold				
goldenrod				
gray				
green				
greenyellow				
grey				
honeydew				
hotpink				
indianred				
indigo				
ivory				
khaki				
lavender				
lavenderblush				
lawngreen				
lemonchiffon				
lightblue				
lightcoral				
lightcyan				
lightgoldenrod				
lightgoldenrodyellow				
lightgray				
lightgreen				
lightgrey				
lightpink				
lightsalmon				
lightseagreen				
lightskyblue				
lightslateblue				
lightslategray				
lightslategrey				
lightsteelblue				
lightyellow				
lime				
limegreen				
linen				
magenta				
maroon				
mediumaquamarine				
mediumblue				
mediumorchid				
mediumpurple				
mediumseagreen				
mediumslateblue				
mediumspringgreen				
mediumturquoise				
mediumvioletred				
midnightblue				
mintcream				
mistyrose				
moccasin				

	1	2	3	4
navajowhite				
navy				
navyblue				
oldlace				
olive				
olivedrab				
orange				
orangered				
orchid				
palegoldenrod				
palegreen				
paleturquoise				
palevioletred				
papayawhip				
peachpuff				
peru				
pink				
plum				
powderblue				
purple				
red				
rosybrown				
royalblue				
saddlebrown				
salmon				
sandybrown				
seagreen				
seashell				
sienna				
silver				
skyblue				
slateblue				
slategray				
slategrey				
snow				
springgreen				
steelblue				
tan				
teal				
thistle				
tomato				
turquoise				
violet				
violetred				
wheat				
white				
whitesmoke				
yellow				
yellowgreen				

ÉCRIRE UNE COMMANDE PYTHON SUR PLUSIEURS LIGNES

- Utiliser la continuité implicite des lignes au sein des parenthèses/crochets/accolades
- Utiliser en dernier recours le backslash "\" (= line break)

continuité implicite	backslash
`def __init__(self, a, b, c,` `  d, e, f, g):`	
`output = (a + b + c` `  + d + e + f)`	`output = a + b + c \` `  + d + e + f`
`lst = [a, b, c,` `  d, e, f]`	
`if (a > 5` `  and a < 10):`	`if a > 5 \` `  and a < 10:`