

CNESC Informatique

Python
Reference Booklet
2019–2020

Version 2.2.1 – Luxembourg, le 13 mars 2020

Table des matières

1	Notions de base	2
1.1	Types de valeurs	2
1.2	Différents opérateurs	2
1.2.1	Opérateurs d'affectation	2
1.2.2	Opérateurs mathématiques	2
1.2.3	Opérateurs de comparaison	2
1.2.4	Opérateurs logiques	3
1.2.5	Priorité des opérateurs	3
1.3	Divers	3
1.3.1	Commentaires	3
1.3.2	Conversion de types	3
1.3.3	Entrée de données	3
1.3.4	Affichage dans la console	4
1.3.5	Importations	4
2	Structure alternative	5
2.1	Différentes syntaxes	5
2.1.1	Syntaxe simplifiée	5
2.1.2	Syntaxe complète	5
2.1.3	Syntaxe avancée	5
3	Boucles	6
3.1	For	6
3.2	While	6
4	Listes	7
4.1	Création et utilisation des listes	7
4.2	Création automatisée de listes	8
4.3	Copier une liste	8
4.4	Sous-listes	8
5	Strings	9
5.1	Notions de base	9
5.2	Création et utilisation des strings	9
5.3	Méthodes utiles	10
6	Fonctions	11
7	Autres types de données	12
7.1	Tuplets	12
7.2	Dictionnaires	12
8	Mathématiques	13
8.1	Fonctions mathématiques : math package	13
8.2	Nombres pseudo-aléatoires : random package	13
9	Classes - Programmation orientée objet	14
10	PyGame	15
10.1	Structure d'un programme Pygame	15
10.2	Éléments graphiques	15
10.2.1	Utiliser des couleurs	15
10.2.2	Tracer une ligne	15
10.2.3	Tracer un rectangle	16
10.2.4	Tracer une ellipse	16
10.2.5	Tracer un cercle	16
10.2.6	Écrire un texte dans une fenêtre	16
10.3	Gérer les événements	17
10.3.1	Interaction avec la fenêtre	17
10.3.2	Clavier	17
10.3.3	Souris	18
11	Pillow	19

1 Notions de base

1.1 Types de valeurs

type		exemple
<code>int</code>	entier	<code>a = 5</code>
<code>float</code>	virgule flottante	<code>c = 5.6</code>
<code>complex</code>	nombre complexe	<code>d = 5 + 4j</code>
<code>str</code>	chaîne de caractères	<code>e = "hello"</code>
<code>list</code>	liste	<code>my_list = [23, 45]</code>
<code>tuple</code>	tuplet	<code>my_tuple = ("a", 2.4, 45, "hello")</code>
<code>dict</code>	dictionnaire	<code>my_dict = {"name": "John", "age": 42}</code>

1.2 Différents opérateurs

1.2.1 Opérateurs d'affectation

```
x = 42          # simple assignment
x = y = z = 42 # multiple assignment
x, y = 42, 0.3 # parallel assignment
```

1.2.2 Opérateurs mathématiques

symbole	effet	exemple	result
<code>+</code>	addition	<code>6 + 4</code>	10
<code>-</code>	soustraction	<code>6 - 4</code>	2
<code>*</code>	multiplication	<code>6 * 4</code>	24
<code>/</code>	division	<code>6 / 4</code>	1.5
<code>**</code>	puissance	<code>6 ** 4</code>	1296
<code>//</code>	quotient de la division entière	<code>6 // 4</code>	1
		<code>-6.5 // 4.1</code>	-2.0
<code>%</code>	reste positif de la div. entière	<code>6 % 4</code>	2
		<code>-6.5 % 4.1</code>	1.7

1.2.3 Opérateurs de comparaison

symbole	effet
<code><</code>	strictement inférieur
<code>></code>	strictement supérieur
<code><=</code>	inférieur ou égal
<code>>=</code>	supérieur ou égal
<code>==</code>	égal
<code>!=</code>	différent de

1.2.4 Opérateurs logiques

X	Y	X and Y	X or Y
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

X	not X
False	True
True	False

1.2.5 Priorité des opérateurs

1	**	puissance
2	-, +	signe
3	*, /, //, %	multiplication et division
4	+, -	addition et soustraction
5	==, <=, >=, <, >, !=	opérateurs de comparaison
6	not	logique : not
7	and	logique : and
8	or	logique : or

1.3 Divers

1.3.1 Commentaires

```
# voici un commentaire d'une ligne
```

```
'''commentaire_sur  
plusieurs_lignes  
dans_le_code_source  
'''
```

```
"""commentaire_sur  
plusieurs_lignes  
dans_le_code_source  
"""
```

1.3.2 Conversion de types

```
int(s) # convert string s to integer  
float(s) # convert string to float  
str(n) # convert integer or float n to string  
list(x) # convert tuple, range or similar to list
```

1.3.3 Entrée de données

```
# Enter a string without prompt  
s = input()  
  
# Enter a string with prompt  
t = input("Enter a string: ")  
  
# Enter an integer with prompt (error if bad entry)  
n = int(input("Enter an integer number: "))  
  
# Enter a float with prompt (error if bad entry)  
m = float(input("Enter a float number: "))
```

1.3.4 Affichage dans la console

```
# print nothing, and new line
print()

# the string 'Hello' is printed
print("Hello")

# print both strings with a blank space 'Hello World'
print("Hello", "World")

# concatenate and print as one string 'HelloWorld'
print("Hello" + "World")

# print a string and a number (two separate strings) 'Hello 42'
print("Hello", 42)

# print a string and a number (with conversion) 'Hello42'
print("Hello" + str(42))

# print the value from a variable
my_number = 42
print("My_number_is", my_number)    # 'My number is 42'

# print a string several times
print(3 * "Hello")
print("Hello" * 3)

# replace 'new line' by any other character(s)
print("We", end = "\u003C3\u003E")
print("Python")
```

1.3.5 Importations

```
# import a package
import math          # recommended

# or
from math import *  # not recommended (many useless identifiers)

# import only what you really need
from math import sqrt, cos, sin, pi
```

2 Structure alternative

2.1 Différentes syntaxes

2.1.1 Syntaxe simplifiée

```
if <condition>:  
    <instruction(s)>
```

2.1.2 Syntaxe complète

```
if <condition>:  
    <instruction(s)>  
else:  
    <instruction(s)>
```

Opérateur ternaire :

```
my_var = <expr_if_true> if <condition> else <expr_if_false>
```

2.1.3 Syntaxe avancée

```
if <condition_1>:  
    <instruction(s)>  
elif <condition_2>:  
    <instruction(s)>  
...  
elif <condition_n>:  
    <instruction(s)>  
else:  
    <instruction(s)>
```

3 Boucles

3.1 For

```
for <iterator> in <list_of_values>:  
    <instruction(s)>
```

On utilise l'expression : `range(start, stop, step)`

```
for i in range(10):  
    print(i)                # values 0, 1, ..., 9  
  
for i in range(3, 8):      # values 3, 4, 5, 6, 7  
    print(i)  
  
for i in range(2, 17, 3):  # values 2, 5, 8, 11, 14  
    print(i)  
  
for i in range(4, 0, -1):  # values 4, 3, 2, 1  
    print(i)  
  
for letter in "Hello!":    # values 'H', 'e', 'l', 'l', 'o', '!'  
    print(letter)  
  
for _ in range(5):        # recommended name  
    print("Hello!")       # iterator not used in loop
```

3.2 While

```
while <condition(s)>:  
    <instruction(s)>  
    # don't forget to update the condition(s)
```

4 Listes

4.1 Création et utilisation des listes

```
# create new empty list
my_list = []

# create list with 2 elements
my_list = ["hello", "world"]

# append element to list
my_list.append(42)

# show list contents (implicit str() conversion)
print(my_list)

# show specific element in list
print(my_list[0])

# modify specific element in list
my_list[0] = "goodbye"

# count element in list
n = my_list.count("hello")

# verify if element is in list
is_in_list = 42 in my_list

# find index of first appearance of element (exception ValueError if not found)
n = my_list.index("hello")

# length of list : number of elements
n = len(my_list)

# delete element on specific position
del my_list[0]
del(my_list[0])

# delete first occurrence of specific element (error if not found)
my_list.remove(42)
my_list.remove("hello")

# reverse order of list elements
new_list = reversed(my_list)      # my_list is left unchanged
my_list.reverse()                 # in situ

# sort list
new_list = sorted(my_list)        # my_list is left unchanged
my_list.sort()                    # in situ

# concatenate multiple lists
my_list_3 = my_list_1 + my_list_2

# return and remove last element in list (error if empty list)
element = my_list.pop()

# insert element at specific location (index)
my_list.insert(3, "hello")
```


4.2 Création automatisée de listes

```
my_list = list(range(6))          # [0, 1, 2, 3, 4, 5]
my_list = list(range(8, -4, -3))  # [8, 5, 2, -1]
```

4.3 Copier une liste

```
my_list = [2, 3, 4]
copy_list = my_list[:]           # first-level copy (shallow copy)
matrix = [[1, 2], [3, 4]]       # list of lists
copy_matrix = [x[:] for x in matrix] # second-level copy
from copy import deepcopy
copy_list = deepcopy(my_list)    # deep copy
```

4.4 Sous-listes

```
my_list = ["a", "b", "c", "d", "e"]
print(my_list[1:3])  # ['b', 'c']
print(my_list[:2])   # ['a', 'b']
print(my_list[3:])   # ['d', 'e']
print(my_list[2:-1]) # ['c', 'd']
```

5 Strings

Attention : Les chaînes de caractères (nommées par la suite strings) ne peuvent être changées après leur création. On peut accéder aux différents caractères, mais on ne peut pas supprimer un caractère (ou une sous-chaîne) dans la chaîne originale. Pour modifier une chaîne il faut donc se faire une copie qui contient les changements. Une espace dans un string est aussi traitée comme un caractère (blank character).

5.1 Notions de base

```
my_string_1 = 'hello'
my_string_2 = "world"
my_string_3 = """first_row_of_this
string_that_spans_over
three_rows"""
my_string_4 = "mu" + 5 * "ha" + 2 * "!"
```

5.2 Création et utilisation des strings

```
# create new string
my_string_1 = "hello_world"

# create a copy of a string
my_string_2 = my_string_1

# access specific character
print(my_string_1[3])

# access specific character (starting from end of string)
# -1 : last character, -2 : second last character, etc.
print(my_string_1[-1])

# number of characters in string (length of string)
n = len(my_string_1)

# create substring
new_string = my_string_1[2:6]
new_string = my_string_1[4:-2]
new_string = my_string_1[:3]
new_string = my_string_1[2:]

# index of first occurrence of element (error if not found)
n = my_string_1.index("_")
new_string = my_string_1[n+1:]

# concatenation of several strings
new_string = my_string_1 + "_and_a_" + "number_" + str(42)
```

5.3 Méthodes utiles

```
s1 = "my_TINY_text."          # example text

s2 = s1.capitalize()         # 'My tiny text.'
s2 = s1.lower()              # 'my tiny text.'
s2 = s1.upper()              # 'MY TINY TEXT.'

# strip() removes leading/trailing spaces or characters
s2 = "  text  ".strip()      # 'text'
s2 = s1.strip("met.")        # 'y TINY tex'

li = s1.split()              # ['my', 'TINY', 'text.']
li = s1.split("t")          # ['my TINY ', 'ex', '.']

n = s1.find("TI")            # 3 (index, -1 if not found)
n = s1.find("t", 9, -1)      # 11 (looks only at s1[9:-1])
s2 = "abababa".replace("ab", "c") # 'ccca'
s2 = "abababa".replace("a", "c", 2) # 'cbcbaba' (first 2 occurrences replaced)

# test methods (return True or False)
my_string.isalpha()         # not empty and only letters?
my_string.isdigit()         # not empty and only digits 0...9 ?
my_string.isalnum()         # not empty and only letters and digits?
my_string.islower()         # not empty and no uppercase letter?
my_string.isupper()         # not empty and no lowercase letter?
my_string.isspace()         # not empty and only (white)spaces?
```

6 Fonctions

```
def <name of function>(param_1, ..., param_n):  
    <instruction(s)>
```

```
def <name of function>(param_1, ..., param_n):  
    <instruction(s)>  
    return <answer>
```

```
def <name of function>():  
    <instruction(s)>  
    return <answer>
```

```
def <name of function>(param_1, ..., param_m, param_n=42):  
    <instruction(s)>  
    return <answer>
```

```
# example call of function in main part  
my_result = my_function(arg_1, ..., arg_n)
```

7 Autres types de données

7.1 Tuplets

```
# examples of tuples
t1 = (10, 20, 30)
# or
t1 = 10, 20, 30

(x, y, z) = t1
# or
x, y, z = t1
```

7.2 Dictionnaires

```
# examples of dict
my_dict = { "M": 1000, "D": 500, "C": 100 }
my_dict["L"] = 50 # new entry or modification in dict
del my_dict["L"] # removes item from dict (error if unknown)

my_value = my_dict["M"] # value of corresponding key (error if unknown)
my_value = my_dict.get("B", 0) # default value if key unknown

n = len(my_dict) # size
list_of_keys = my_dict.keys() # not sorted
list_of_keys = list(my_dict) # idem
sorted_list_of_keys = sorted(my_dict)
list_of_values = my_dict.values() # not sorted
list_of_items = my_dict.items() # item = (key, value)

"M" in my_dict # True (key has been found)
"D" not in my_dict # False (key has been found)

for k in my_dict:
    <instruction(s)> # k loops through the keys of the dict
for k, v in my_dict.items():
    <instruction(s)> # (k, v) loops through the (key, value) couples
```

8 Mathématiques

8.1 Fonctions mathématiques : math package

```
# import some math functions
# abs(), round() always available without import
from math import sin, cos, tan, pi, sqrt
from math import ceil, trunc, floor
x = cos(pi)      # -1.0
x = sin(30)     # -0.9880316240928618 (arg in radians!)
x = abs(-3)     # 3
x = sqrt(256)   # 16
x = ceil(1.23)  # 2 (smallest integer larger or equal)
x = trunc(-8.76) # -8 (strip fractional part)
x = floor(-8.76) # -9 (largest integer smaller or equal)
x = round(4.6)  # 5 (rounds to nearest integer)
x = round(3.5)  # 4 (rounds to nearest EVEN integer!!)
x = round(4.5)  # 4 (rounds to nearest EVEN integer!!)
```

8.2 Nombres pseudo-aléatoires : random package

```
# import some random functions
from random import random, randint, randrange
x = random()    # random float 0.0 <= x < 1.0
x = randrange(6) # random int 0 <= x < 6
x = randrange(1, 6) # random int 1 <= x < 6
x = randint(1, 6) # random int 1 <= x <= 6
```

9 Classes - Programmation orientée objet

```
# example class
class My_class: # defines a new class
    def __init__(self, param1 = def1, param2 = def2):
        # initializer with 2 parameters and default values
        self.var1 = param1 # attribute of class instance
        self.var2 = param2 # all attributes are public!
    def my_method_1(self):
        # method callable on every class instance
        <instruction(s)>
        return my_result # optional
    def my_method_2(self, param1, param2):
        # method callable with two additional arguments
        <instruction(s)>
        return my_result # optional

# example: use of My_class
obj = My_class(arg1, arg2) # calls init with 2 args
obj = My_class(arg1) # default value for 2nd parameter
obj = My_class() # default values for both parameters

obj.my_method_1() # calls method with self = obj
obj.my_method_2(arg1, arg2) # calls method with self = obj and 2 more args

print(obj.var1) # direct access to public attribute
obj.var2 = <expression> # idem (allowed, but not recommended)
```

10 PyGame

10.1 Structure d'un programme Pygame

```
import pygame, sys
from pygame.locals import *

pygame.init()
size = (400, 300)
screen = pygame.display.set_mode(size)
pygame.display.set_caption("Hello_world!")
screen.fill(Color("white"))

FPS = 30
clock = pygame.time.Clock()

done = False
while not done:
    for event in pygame.event.get():
        if event.type == QUIT:
            done = True
            # elif event.type == # check for user interaction
            # do some cool stuff
        pygame.display.update()
        clock.tick(FPS)

pygame.quit()
sys.exit()
```

10.2 Éléments graphiques

10.2.1 Utiliser des couleurs

```
# colors by name
Color("black")
Color("white")

# colors by RGB mode
Color(0,0,0)
Color(255, 255, 255)
```

10.2.2 Tracer une ligne

```
# Parameters for pygame.draw.line:
# surface, color, start_point, end_point[, width]

pygame.draw.line(screen, Color("red"), (0, 0), (10, 10), 5)
```


10.2.3 Tracer un rectangle

```
# Parameters for pygame.draw.rect:  
# surface, color, (x, y, width, height)[, width]  
# width = 0 means rectangle is filled  
  
pygame.draw.rect(screen, Color("red"), (0, 0, 10, 40), 1)
```

10.2.4 Tracer une ellipse

```
# Parameters for pygame.draw.ellipse:  
# surface, color, (x, y, width, height)[, width]  
# width = 0 means ellipse is filled  
  
pygame.draw.ellipse(screen, Color("red"), (0, 0, 10, 40), 1)
```

10.2.5 Tracer un cercle

```
# Parameters for pygame.draw.circle:  
# surface, color, center_point, radius[, width]  
# width = 0 means circle is filled (disc)  
  
pygame.draw.circle(screen, Color("red"), (50, 50), 15, 1)
```

10.2.6 Écrire un texte dans une fenêtre

Exemple : le texte « Hello World! » est écrit en couleur rose au milieu de la fenêtre.

```
font = pygame.font.SysFont("Palatino", 72)  
text = font.render("Hello World!", True, Color("pink"))  
screen.blit(text, ((w - text.get_width()) // 2, (h - text.get_height()) // 2))
```

10.3 Gérer les événements

10.3.1 Interaction avec la fenêtre

```
if event.type == pygame.QUIT:
    # window specific command quit
    <instruction(s)>
```

10.3.2 Clavier

```
if event.type == KEYDOWN:
    # key was pressed
    if event.key == K_<name of key>:
        <instruction(s)>
```

```
if event.type == KEYUP:
    # key was released
    if event.key == K_<name of key>:
        <instruction(s)>
```

Exemples :

```
if event.type == KEYDOWN:
    if event.key == K_SPACE:
        print("SPACE_pressed")
    elif event.key == K_b:
        print("b_key_pressed")
    elif event.key == K_LEFT:
        print("left_arrow_key_pressed")
```

event.key Constantes :

```
K_a, K_b, ...           # a..z
K_0, K_1, ...           # 0..9
K_KP0, K_KP1, ...      # keypad 0..9
K_UP, K_DOWN, K_LEFT, K_RIGHT,
K_LALT, K_RSHIFT, K_LCTRL, ...
K_SPACE, K_RETURN, K_ESCAPE, ...
...                     # other keys
```

10.3.3 Souris

```
if event.type == MOUSEBUTTONDOWN:
    if pygame.mouse.get_pressed() == (True, False, False):
        # left mouse button is down
        <instruction(s)>
    if pygame.mouse.get_pressed() == (False, False, True):
        # right mouse button is down
        <instruction(s)>
    if pygame.mouse.get_pressed() == (False, True, False):
        # middle mouse button is down
        <instruction(s)>
```

```
if event.type == MOUSEBUTTONUP:
    if pygame.mouse.get_pressed() == (False, False, False):
        # all mouse buttons are released
        <instruction(s)>
    if pygame.mouse.get_pressed() == (True, False, False):
        # left mouse button is still down
        <instruction(s)>
    if pygame.mouse.get_pressed() == (False, False, True):
        # right mouse button is still down
        <instruction(s)>
    if pygame.mouse.get_pressed() == (False, True, False):
        # middle mouse button is still down
        <instruction(s)>
```

```
if event.type == MOUSEMOTION:
    <instruction(s)>
```

Position de la souris :

```
# save (x, y) position where mouse pointer is during event
(x, y) = pygame.mouse.get_pos()
```

11 Pillow

```
# import package and open image :
from PIL import Image
imgFrog = Image.open("frog.jpg")

# show image (default photo viewer) :
imgFrog.show()

# save image in specific format(bmp, gif, jpg, png, ...):
imgFrog.save("frog.png")

# crop image :
corners = (600, 150, 1400, 750)
imgRegion = imgFrog.crop(corners)

# copy and paste image into another image at position (x, y) :
imgClone = imgFrog.copy()
imgLake.paste(imgClone, (x, y))

# split image in color bands :
r, g, b = imgFrog.split()

# recompose / merge image from splitted bands :
imgNewFrog = Image.merge("RGB", (r,g,b))

# convert image to grayscale :
imgGrayFrog = imgFrog.convert("L")

# resize image :
size = (400, 300)
imgSmallFrog = imgFrog.resize(size)

# transpose, rotate, filter :
imgResult = imgFrog.transpose(Image.ROTATE_90)
imgResult = imgFrog.rotate(45)
imgResult = imgFrog.transpose(Image.FLIP_TOP_BOTTOM)
from PIL import ImageFilter
imgResult = imgFrog.filter(ImageFilter.SHARPEN)

# create new white image :
imgNew = Image.new("RGB", (800,600), (255, 255, 255))
# create new monoband black image
imgHistogram = Image.new("L", (256, 500), 0)

# access specific pixels :
pixels = imgFrog.load()
value = pixels[x,y]
pixels[0,0] = value // 2
```

VALEURS ET TYPES

Types numériques

int	integer (32-bit), entier compris entre -2147483648 ... 2147483647
long	long integer, entier compris entre -∞ ... +∞ (précision illimitée)
float	floating point number, nombre avec point décimal

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

str	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

Noms des variables

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)

- Il y a une différence entre caractères majuscules et minuscules (case sensitive)

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

CHAINES DE CARACTERES (= SEQUENCES NON-MODIFIABLES)

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

String literals

ord('A')	↪ return integer Unicode code point for char (e.g. 65)
chr(65)	↪ return string representing char at that point (e.g. 'A')
"texte" ou 'texte'	délimiteurs doivent être identiques
""" chaîne sur plusieurs lignes """	chaîne sur plusieurs lignes, délimitée par """" ou '''
"abc\ndef" ou 'abc\ndef'	inclure le délimiteur dans la chaîne
\n	passage à la ligne suivante
\\	pour afficher \

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 numéroté)

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
{var1:3d} ↪ display as integer, padding = 3	{:.2f} ↪ display as float with 2 decimals
	{: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()	renvoie une copie en lettres minuscules
s.upper()	renvoie une copie en lettres 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 une 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

LISTES (= SEQUENCES MODIFIABLES) -> []

type: list

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

Création

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

Remarque

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

list comprehensions (computed lists)

lst = [expr for var in sequence] expr is evaluated once for every item in sequence

Exemple: création d'une matrice 3x3

p = [x[:] for x in [[0]*3]*3]	ou	on construit d'abord 3 vecteurs composés chacun de 3 composants nuls, le résultat (x) est copié dans p, pour que les 3 vecteurs-lignes obtenus deviennent indépendants et ne pointent pas sur le même objet
p = [[0,0,0], [0,0,0], [0,0,0]]		

Affichage

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) (seuls les éléments avec index = multiple de step inclus)
lst[start:end:step]	

Exemples

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 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 variables pointent alors sur la même liste)
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)

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) ou lst += [item1, ..., item_n]	ajoute un élément à une liste
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
lst.pop(index)	(à la position indiquée par index)
lst.reverse()	inverse les items d'une liste
lst.sort()	trier la liste (modifie la liste)
lst.insert(index, item)	insère l'item à la position donnée par index

Attention:

lst = [1, 2, 3, 4]	lst = [1, 2, 3, 4]
lst[2] = [7,8,9] >>> [1, 2, [7, 8, 9], 4]	lst[2:2] = [7,8,9] >>> [1, 2, 7, 8, 9, 4]
(liste imbriquée)	(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
lst.find(item)	retourne l'index de la 1ère occurrence de item, sinon retourne -1
item in lst (item not in lst)	indique si 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 i in range(len(lst)-1, -1, -1):	effacer certains éléments d'une liste
... code pour effacer des items	↪ il faut parcourir la liste de la fin au début, si on a besoin de l'index
while i < len(lst):	effacer certains éléments d'une liste
if ... code pour effacer items	
else:	
i = i + 1	
if lst: ou if len(lst) > 0:	test si la liste lst n'est pas vide

RANGE (= SEQUENCES NON MODIFIABLES)

Retourne une séquence non modifiable d'entiers

range([start, stop[, step]])	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]	
(start, stop, step = integers)	range(0, -10, -2) ↪ [0, -2, 4, -6, -8]

LES TUPLETS (TUPLES) -> ()

type: tuple

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

Création

<code>tuple = (a, b, c, ...)</code>	créer un tuplet
<code>tuple = a, b, c, ...</code>	(on peut omettre les parenthèses, si clair)
<code>tuple1 = tuple2</code>	copier un tuplet

Extraction

<code>(x, y, z) = tuple</code> ou <code>x, y, z = tuple</code>	extraire les éléments d'un tuplet
--	-----------------------------------

Affichage -> voir listes

Premier élément d'un tuplet ↪ index 0

<code>tuple[index]</code>	retourne l'élément à la position <code>index</code> (un <code>index < 0</code> ↪ accède aux éléments à partir de la fin)
<code>tuple[start:end]</code>	retourne une sous-liste de l'indice <code>[start ; end[</code>

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

<code>dic = {}</code> ou <code>dic = dic()</code>	créer un dictionnaire vide
<code>dic = {key1: val1, key2: val2, ..}</code>	créer un dictionnaire déjà rempli
<code>dic[key] = value</code>	ajouter une clé : <i>valeur</i> au dictionnaire si la clé n'existe pas encore, sinon elle est remplacée

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

Affichage

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

Modification

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

Divers

<code>len(dic)</code>	renvoie le nombre d'éléments dans le dictionnaire
<code>if key in dic</code> , <code>if key not in dic</code>	tester si le dictionnaire contient une certaine clé
<code>for c in dic.keys():</code> ou <code>for c in dic:</code>	parcourir les clés d'un dictionnaire
<code>for c, v in dic.items():</code>	parcourir les éléments du dictionnaire
<code>copie = dic.copy()</code>	créer une copie (shallow copy) du dictionnaire (une affectation crée seulement un nouveau pointeur sur le même dictionnaire) - 1st level copy
<code>copie = copy.deepcopy(dic)</code>	import copy (any level copy)
<code>max(dic, key=len)</code>	retourne la clé la plus longue

EXPRESSIONS ET OPERATEURS

Opérateurs entourés d'espaces.

Utiliser des parenthèses pour grouper des opérations (modifier la priorité)

Opérateurs mathématiques

La 1^{ère} colonne indique la priorité des opérateurs

1.	**	exponentiation	
2.	-, +	signe	
3.	*	multiplication	<code>x * 3</code> ↪ <code>x = x * 3</code>
	/	division (entière ou réelle)	<code>x / 3</code> ↪ <code>x = x / 3</code>
	//	quotient de la division entière	
	%	modulo, reste (positif) de la division entière	
4.	+	addition	<code>x += 3</code> ↪ <code>x = x + 3</code>
	-	soustraction	<code>x -= 3</code> ↪ <code>x = x - 3</code>

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: <code>x >= a</code> ou <code>b >= x >= a</code> pour <code>a <= b</code>)
	<=	inférieur ou égal à (exemple: <code>x <= b</code> ou <code>a <= x <= b</code>)

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)
8.	x or y	ou (retourne y, si x est faux, sinon x)

and ne vérifie le 2^e argument que si le 1^{er} argument est vrai
or ne vérifie le 2^e argument que si le 1^{er} argument est faux

Affectation

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

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

ENTREE / SORTIE

Entrée

<code>var = input()</code>	renvoie une chaîne de caractères
<code>var = input(message)</code>	renvoie une chaîne de caractères et affiche le message
<code>int = int(input(...))</code>	renvoie un entier
<code>float = float(input(...))</code>	renvoie un nombre décimal

Sortie

<code>print(text, end="final")</code>	affiche <code>text</code> et termine avec <code>final</code> (par défaut <code>end="\n"</code>)
<code>print("abc", "def")</code>	↪ <code>abc def</code> (arguments séparés par une espace, nouvelle ligne)
<code>print("abc", end="+")</code>	↪ <code>abc+</code> (pas de passage à la ligne)
<code>print(var)</code>	<code>var</code> est convertit en chaîne et affichée
<code>print()</code>	simple passage à la ligne
<code>print(str * n) print(n * str)</code>	afficher <code>n</code> fois le texte <code>str</code>

LES COMMENTAIRES

<code># commentaire</code>	sur une seule ligne
<code>"""comments"""</code> ou <code>"""'comments'"""</code>	sur plusieurs lignes (= string literal)

STRUCTURE ALTERNATIVE ET RÉPÉTITIVE

Structure alternative

<code>if condition1:</code> <code>instruction(s)</code>	<ul style="list-style-type: none"> exécute seulement les instructions, où la condition est vérifiée si aucune condition est vérifiée, les instructions de else sont exécutées else et elif sont optionnels
<code>elif condition2:</code> <code>instructions(s)</code> ...	
<code>else:</code> <code>instruction(s)</code>	
<code><on true> if <expr> else <on false></code>	<ul style="list-style-type: none"> ternary operator

Structure répétitive (boucle for)

<code>for itérateur in liste de valeurs:</code> <code>instruction(s)</code>	<ul style="list-style-type: none"> répète les instructions pour chaque élément de la liste nombre de répétitions = connu au départ
--	--

Structure répétitive (boucle while)

<code>while condition:</code> <code>instruction(s)</code>	<ul style="list-style-type: none"> 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 break ↪ quitte la boucle immédiatement
--	---

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, tuplets) ↪ passage par valeur (valeurs copiés)
- arguments complexes (listes, dictionnaires) => passage par référence (vers les originaux)

Définition et appel

<code>def my_function(par1, ..., par_n):</code> <code>instruction(s)</code> ... <code>return var</code>	<p>définit une fonction <code>my_function</code></p> <ul style="list-style-type: none"> <code>par1 .. par_n</code> sont les paramètres une ou plusieurs instructions return... peut renvoyer plusieurs réponses (tuplet, liste) <p>Si la fonction ne contient pas d'instruction return, la valeur None est renvoyée</p>
<code>my_function(arg1, ... arg_n)</code> <code>var = my_function(arg1, ... arg_n)</code>	appel de la fonction, arguments affectés aux paramètres dans le même ordre d'apparition
<code>def func(par1, ..., par_n = val):</code> ex: <code>def add(elem, to = None):</code> <code>if to is None:</code> <code>to = []</code>	<p>paramètre par défaut</p> <p>ATTENTION: def add(elem, to = []): does not work, because python default args, are only evaluated once, and used for all function calls</p>
<code>def func(par1, ..., *par_n):</code>	*par_n = nombre variable de paramètres (liste)

<https://docs.python-guide.org/writing/gotchas/>

Variables globales

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

<code>def func(...):</code> <code>global var</code>	<code>var</code> est déclaré comme variable globale, la variable <code>var</code> à l'extérieur de la boucle est modifiée également
--	---

UTILISATION DE MODULES (BIBLIOTHÈQUES)

Utiliser des modules

<code>import module</code>	importe tout le module, il faut préfixer par le nom du module ex: <code>import math</code> ↪ <code>math.sqrt()</code>
<code>import module as name</code>	
<code>from module import *</code> *** A EVITER ***	intègre toutes les méthodes de <i>module</i> , pas besoin de préfixer le nom du module ex: <code>from math import *</code> ↪ <code>sqrt()</code>
<code>from module import m1, m2, ..</code>	intègre seulement les méthodes mentionnées
<code>from math import sqrt, cos</code>	↪ <code>sqrt(...)</code> , <code>cos(...)</code>

MODULE : MATH

import math

<code>math.pi</code>	le nombre pi
<code>math.cos(x) / .sin(x) / .tan(x)</code>	cosinus/sinus/tangente d'un angle en radian
<code>math.sqrt(x)</code>	racine carrée
<code>math.abs(x)</code>	valeur absolue (aussi nombres complexes)
<code>math.fabs(x)</code>	valeur absolue ↪ retourne un float
<code>math.ceil(x)</code>	x est arrondi vers le haut
<code>math.floor(x)</code>	x est arrondi vers le bas
<code>math.trunc(x)</code>	retourne l'entier sans partie décimale
<code>math.round(x)</code>	x est arrondi vers l'entier le plus proche • <code>round(3.5)</code> ↪ 4 (rounds to nearest EVEN integer) • <code>round(4.5)</code> ↪ 4 (rounds to nearest EVEN integer)
<code>math.pow(x, y)</code>	x exposant y

MODULE : RANDOM

import random

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

MODULE : SYS

import sys

<code>sys.stdin.readline()</code>	lit la prochaine ligne de STDIN (" si EOF)
<code>sys.maxsize</code>	valeur max. d'un entier en Python (32-bit ↪ 2 ³¹ , 64-bit ↪ 2 ⁶³)

MODULE : TIME

import time

<code>t1_start = time.process_time()</code>	Return process time of current process as float in seconds
<code>t1_stop = time.process_time()</code> <code>print(t1_stop - t1_start)</code>	

LES FICHIERS

Entrées/sorties 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)
<code>command > filename</code>	rediriger la sortie standard vers un fichier (créé/remplacé)
<code>command >> filename</code>	rediriger la sortie standard vers un fichier (ajouté)
<code>command > NUL</code>	annuler sortie vers STDOUT
<code>command < filename</code>	rediriger entrée depuis un fichier

Tubes et filtres

<code>command1 command2</code>	rediriger la sortie de <i>command1</i> comme entrée à <i>command2</i>
----------------------------------	---

Manipulation de fichiers

<code>file_object = open(file, mode='r')</code>	retourne un objet fichier
<code>file_object.readline()</code>	retourne la prochaine ligne complète avec caractère fin de ligne (retourne une chaîne vide " si la fin du fichier est atteint)
<code>file_object.write(str)</code>	écrit dans <i>file_object</i> la chaîne <i>str</i>
<code>file_object.close()</code>	fermer le <i>file_object</i> (si traitement du fichier est terminé)

Valeurs pour mode

'r'	mode lecture
'w'	mode écriture
'a'	mode écriture/ajout (à la fin)

Lire de STDIN en Python (manière de filtres)

<code>import sys</code> <code>line = sys.stdin.readline()</code> <code>while line != "":</code> ... <code>line = sys.stdin.readline()</code>	lire les données de STDIN (ou) <code>import sys</code> <code>for line in sys.stdin:</code> ...
--	---

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

MODULES ET LIBRAIRIES (PACKAGES)

Modules

↪ fichiers dans lesquels on regroupe différentes fonctions

- créer un fichier (*module*) contenant des fonctions ↪ utiliser les fonctions du module
- dans un 2e fichier utiliser: `import 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`

- créer un dossier ↪ créer une librairie
- ajouter des modules
- créer le fichier vide `__init__.py` dans le dossier

Installer des librairies (packages) externes

PyCharm

↪ File → Settings → Project: votre projet actuel

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

↪ Choisir librairie 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/>)

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

PROGRAMMATION ORIENTE OBJET (POO)

objet oriented programming (OOP)

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

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)

<code>class ClassName:</code>	définit la classe <i>ClassName</i> (CamelCase)
<code>def __init__(self, par1, ..., par_n):</code> <code>self.var1 = ...</code> <code>self.var2 = ...</code>	les fonctions sont appelées méthodes • <code>__init__()</code> ↪ constructeur, appelé lors de l'instanciation
<code>def __str__(self):</code> ... <code>return chaîne_de_texte</code>	• <code>__str__(self)</code> ↪ string representation of object, e.g. <code>print(object)</code> • <code>self</code> doit être le 1 ^{er} paramètre et référencie la classe elle-même
<code>def method(self, ...):</code> ... <code>return result</code>	• <code>self.var_x</code> ↪ attributs, inaccessibles en dehors de l'objet
<code>def _method2(self, ...)</code>	• <code>method1()</code> ↪ accessible à l'extérieur • <code>_method2()</code> ↪ caché, mais accessible à l'extérieur
<code>obj = ClassName(...)</code>	instancie un nouvel objet de la classe dans la mémoire
<code>obj.method(...)</code>	appel de la méthode de l'objet

RECURSIVITE

Algorithme récursif ↪ algorithme qui fait appel(s) à lui-même

Attention: il faut prévoir une condition d'arrêt (= cas de base)

PYGAME

Pygame = bibliothèque pour créer des jeux

Structure d'un programme Pygame

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

Types d'événements

<pre>for event in pygame.event.get(): if event.type == QUIT: done = True elif event.type == <type d'événement>: <instruction(s)> ...</pre>	Gestion de tous les événements dans une seule boucle for à l'intérieur de la boucle principale. Toutes les instructions if doivent être regroupées dans une seule boucle for
--	--

Événement de terminaison

QUIT	L'utilisateur a cliqué sur la croix de fermeture de la fenêtre. Pour terminer correctement, utiliser: pygame.quit() sys.exit()
------	--

Événements - clavier

KEYDOWN / KEYUP	une touche du clavier est enfoncée / relâchée
eventkey	indique quelle touche a été enfoncée https://www.pygame.org/docs/ref/key.html
K_a	a (pareil pour le reste de l'alphabet)
K_0	0 en haut (pareil pour les autres chiffres)
K_KP0	0 sur pavé numérique (pareil ...)
K_LALT, K_RALT	touche ALT
K_LSHIFT, K_RSHIFT	touche SHIFT
K_LCTRL, K_RCTRL	touche CONTROL
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)

Événements - souris

MOUSEBUTTONDOWN	un bouton de la souris a été enfoncé
MOUSEBUTTONUP	un bouton de la souris a été relâché
MOUSEMOTION	la souris a été déplacée
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):
pygame.mouse.get_pos()	retourne la position de la souris comme tuple

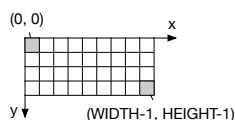
La surface de dessin

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 de tuple



Couleurs (https://en.wikipedia.org/wiki/X11_color_names)

color = 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

Effacer/Remplir surface de dessin

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

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)

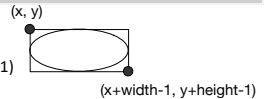
pygame.draw.ellipse(screen, color, bounding_rect, width)	
• bounding_rect = (x, y, width, height) avec x, y = coin supérieur gauche	
• ou rect_tuple = pygame.Rect(x, y, width, height)	
• width = 0 par défaut (= ellipse pleine)	

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()	rafraîchir la surface de dessin pour afficher les dessins
pygame.display.flip()	
pygame.display.flip(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)	créer un nouveau objet Rect, avec left, top = coin supérieur gauche
rect = Rect((left, top), (width, height))	
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
rect.contains(rect2)	retourne True si rect2 est complètement à l'intérieur de rect
rect.collidepoint(x, y)	retourne True si le point donné se trouve à l'intérieur de rect
rect.collidepoint((x, y))	
rect.colliderect(rect2)	retourne True si les 2 rectangles se touchent

Affichage de textes

pygame.font.SysFont(name, size, bold, italic)	créé un objet de type Font à partir des polices système (bold et italic = False par défaut)
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.flip()	met à jour la surface de dessin

Exemple

font = pygame.font.SysFont("comicansms", 20)	créé un objet font
s_text = font.render("Hello", True, Color("green"))	créé nouvelle surface avec texte
screen.blit(s_text, (100, 50))	copie la surface s_text sur screen à la position indiquée et mise à jour
pygame.display.flip()	

ÉCRIRE UNE COMMANDE PYTHON SUR PLUSIEURS LIGNES

Pour 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 "\n" (= 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: