Python 3 Cheat Sheet

by B4LBU via cheatography.com/146469/cs/31689/

commentaire

# commentaire	commentaire simple
""" docstring """	commentaire de type docstring, appellable avec help()

string method

capitalize()	1 lettre en majuscule et reste en minuscule
find(model)	renvoie le plus petit index où model est dans la string
isalnum()	True si string est alphanumérique
isdecimal()	True si string est un décimal
isdigit()	True si strind est en digit
islower()	True si string est en minuscule
isupper()	True si toute la string est en majuscule
'.'.join(itérable)	crée une string à partir d'un itérable et le '.' est le séparateur entre chaque élément de la nouvelle string
lower()	returne un copie de la string en minuscule
upper()	retourne une copie de la string en majuscule
replace(old, new, count=-1)	retourne unen copie de la string avec les old remplacé par new, count correspond au nombre maximum de remplacement. -1 remplace toutes les occurences
split(sep=None)	retourne une list des mots de la string, si sep = '', la liste contient tous les caractères

slicing

str[init:end:step]	init est le début de la liste, par défaut 0, end est la fin voulue et step est le pas, par defaut 1.
str[::-1]	renvoie la string à l'envert

f-string

f"{var} text"	retourne la valeur de la variable accompagnée du text
{:<6} {:>6} {:^6}	aligné à gauche, droite ou centré sur 6 caractères, possibilité d'ajouter un caractère pour remplir les emplacements vides

changement de base

bin()	0b...
oct()	0o...
hex()	0x...
int()	...

vrac & tips

while (var := input()) > ...	permet d'affecter une valeur au sein d'une expression plus large. Le input ainsi que le while est un exemple
$ pydoc3 -w /chemin_du_script	génere un fichier html avec la pydoc
subprocess.run(["commande_bash", argument])	avec import subprocess pour executer une commande bash

fonction

def def (param1, param2, ...):
  commande
  commande
  return val
	definition globale d'une fonction

def f(a, b, /, **kwargs):	avant le / le paramêtre est exclusivement positionnel
var = lambda traitement	la lambda est une fonction anonyme qui ne retourn pas de valeur car le traitement est envoyé dans un variable

regex

import re	module re
re.search(model, string)	renvoie les occurences du model dans la string
re.match(model, str)	cherche si le début de la chaîne correspond
re.fullmatch(model, str)	si l'entièreté de la chaîne correspond

regex (cont)

re.findall(model, str)	ressence toutes les occurances sous la forme d'une liste
re.finditer(regex, str)	produit le même résultat que findall mais sous forme d'un itérateur
re.split(model, str)	créer une liste à partir d'une chaîne de caractères en utilisant un modèle comme séparateur
re.sub(model, elem, str)	remplace le model de str par elem

sauvegarde de donnée

import csv	module csv qui permet de traiter facilement les csv
writer = csv.writer(file)	writer est un objet qui gère l'écriture
writer.writerow(entete)	entête est une liste
writer.writerows(data)	data est une liste de liste (autant de ligne que de liste dans data
reader = csv.reader(file)	itérable permetant la lecture d'un fichier csv. next(reader) permet de sauter une ligne du reader

P.O.O

class Maclasse:	definit ma classe
class B(A):	la classe B hérite de A
def __init__(self, parm ...):	permet de créer une methode d'initialisation de la classe
@classmethod def class_(cls):	methode disponible pour l'ensemble de la classe, cls représente la classe
def instance_(self):	methode applicable uniquement à l'instance, self représente l'instance
if __name__ == '__main__':	les commandes de se blocs sont jouées si le nom main est le nom actuel du programme

P.O.O (cont)

super().__init__(param, ...)	dans le cadre de l'héritage, super() représente le parent au sein de la classe enfant
@property	permet d'accéder à la valeur d'un attribut privé
@<nom_du_getter>.setter	modifier la valeur d'un attribut privé
public : nom / protected : _nom / private : __nom	encapsulation : restreindre l'accès des données entres elles

list method

append()	ajoute un élément à la fin de la liste
clear()	retire tous les éléments de la liste
count(value)	retourne le nombre d'occurence de value
extend(iterable)	agrandit la liste en ajoutant les éléments d'un itérable
insert(index, object)	insert l'objet avant l'index voulu
pop(index=-1)	retire par defaut le dernier élément de la liste. IndexError si la liste est vide
index(value)	retourne le premier index de value. ValueError si value n'est pas dans la liste
remove(value)	retire la première occurance de value. ValueError si elle n'est pas présente
sort(reverse=False)	retourn la liste triée (mais ne modifie pas la liste)

dict method

clear()	retire tous les éléments du dictionnaire
get(key)	retourne la valeur associé à key, sinon None
items()	objet donnant une view du dictionnaire
keys()	view de toutes les keys du dictionnaire
values()	objet donnant une view des valeur du dictionnaire

compréhentions

[var for var in itérable]	crée une nouvelle liste
[int(i) for i in range(21) if i%2 == 0]	exemple avec une liste des 20 premier nombre pair
{k: v for k, v in zip(liste1, liste2)}	dictionnaire (usage du zip)

structure de test

if condition :
  instruction
elif condition :
  instruction
else :
  instruction
	structure classique

valeur = val_vrai if condition else val_faux	structure ternaire, possibilité de chaîner les ternaires

structure de boucle

while condition : instruction	si la condition est remplie, la boucle s'arrête
for var in sequence : instruction	var prendra successivement toutes les valeurs des éléments de sequence. souvent utilisé avec range()
for ... else: while ... else:	la clause else est jouée si le block s'est executé sans erreur

break & continue

break	fin de boucle
continue	fin d'itération (et passe à la suivante)

import

import librairie	importe l'ensemble de la librairie
from librairie import methode	importe uniquement la methode utile (à combiner avec le as)

gestion des erreurs

try:	le code qui suit le bloc try initie potentiellement une erreur
except nameerror:	intercepte l'erreur pour jouer le bloc qui suit
finally:	dans tous les cas joué (erreur ou pas erreur)

gestion des erreurs (cont)

else:	joué si aucune erreur levée
raise nameerror('message')	permet de lever une erreur pour un traitement ulterieur

gestion des fichiers

from pathlib import Path	permet d'écrire Path.methode()
Path.cwd()	répertoire courant
Path.home()	renvoie la home
Path(r'/home/cptain/bin')	crée tout de même l'objet de type fichier même si le chemin n'existe pas
Path.home() .joinpath('bin', 'bash/')	association de chemin

accès au fichier

with p.open() as f: f.methode_Path	avec p un posixPath, cette manière de faire évite de devoir fermer le fichier après le traitement
f.read()	permet la lecture d'un fichier entier
f.readline()	permet de lire une ligne du fichier jusqu'au prochain \n
f.readlines()	créer une liste composée des lignes fi fichier
f.write()	écrire dans un fichier
f.writelines()	écrire à partir d'une liste
f.seek(nb)	positionne le curseur au début si nb=0, à la fin si nb=2
FileNotFoundError	Fichier introuvable (Erreur sur le nom du fichier ou fichier injoignable)
PermissionError	Erreur d'accès (Les droits ugo ou erreur de propriétaire)
IOError	Erreur d'ouverture

méthodes spéciales (les dunders)	
__repr__(self)	affiche une information pour le développeur
__str__(self)	fourrnit un affichage pour l'utilisateur
__ge__(self, other)	>=
__le__(self, other)	<=
__lt__(self, other)	<
__eq__(self, other)	==
__ne__(self, other)	!=
__add__(self, other)	+
__mul__(self, other)	*
__truediv__(self, other)	/
__del__(self)	detruire une objet del()
__len__(self)	Redéfinition de len()