Python Syntaxe :

Mots-clés Python:

Keyword	Description	Exemple du code	Résultat
and	A logical operator ‘et’	x = (5 > 3 and 5 < 10) print(x)	True
as	Pour créer un pseudonyme اسم مستعار	import calendar as c print(c.month_name[1])	January

assert	Pour déboguer لتصحيح الأخطاء	x = "hello"
		
		#if condition returns True, then nothing happens:
		assert x == "hello"
		
		#if condition returns False, AssertionError is raised:
		assert x == "goodbye"
			Traceback (most recent call last):
			  File "demo_ref_keyword_assert.py", line 5, in <module>
			  assert x == "goodbye"
			AssertionError

break	Pour sortir d'une boucle	i = 1
		while i < 9:
		  print(i)
		  if i == 3:
		  break
		  i += 1
			1
			2
			3

class	Pour définir une classe	class Person:
		  name = "John"
		  age = 36
		
		print(Person.name)
			John

continue	Pour continuer à l'itération (التكرار) suivante d'une boucle	for i in range(9):
		  if i == 3:
		  continue
		  print(i)
			1
			2
			4
			5
			6
			7
			8
			9

def	Pour définir une fonction	def my_function(): print("Hello from a function") my_function()	Hello from a function
del	Pour supprimer un objet	x = ["apple", "banana", "cherry"] del x[0] print(x)	['banana', 'cherry']

elif	Utilisé dans les instructions conditionnelles, comme else if	for i in range(-5, 5):
		  if i > 0:
		  print("YES")
		  elif i == 0:
		  print("WHATEVER")
		  else:
		  print("NO")
			NO
			NO
			NO
			NO
			NO
			WHATEVER
			YES
			YES
			YES
			YES

else	Utilisé dans les instructions conditionnelles (si non)	x = 2
		if x > 3:
		  print("YES")
		else:
		  print("NO")
			NO

except	Utilisé avec des exceptions, que faire lorsqu'une exception se produit	# (x > 3) will raise an error because x is a string and 3 is a number, and cannot be compared using a '>':
		
		x = "hello"
		
		try:
		  x > 3
		except NameError:
		  print("You have a variable that is not defined.")
		except TypeError:
		  print("You are comparing values of different type")
			You are comparing values of different type

False	Valeur booléenne, résultat des opérations de comparaison (faux)	print(5 > 6)	False

finally	Utilisé avec des exceptions, un bloc de code qui sera exécuté, qu'il y ait ou non une exception	try:
		  x > 3
		except:
		  print("Something went wrong")
		else:
		  print("Nothing went wrong")
		finally:
		  print("The try...except block is finished")
			Something went wrong
			The try...except block is finished

for	Pour créer une boucle for	for x in range(1, 9): print(x)	1 2 3 4 5 6 7 8
from	Pour importer des éléments spécifiques d'un module	from datetime import time	15:00:00

		x = time(hour=15) print(x)	

global	Pour déclarer une variable globale	#create a function:
		def myfunction():
		  global x
		  x = "hello"
		
		#execute the function:
		myfunction()
		
		#x should now be global, and accessible in the global scope.
		print(x)
			hello

if	Pour faire une instruction conditionnelle	x = 5
		if x > 6:
		  print("YES")
		else:
		  print("NO")
			NO

import	Pour importer un module	import datetime x = datetime.datetime.now() print(x)	2023-04-10 01:53:18.336955
in	To check if a value is present in a list, tuple, etc.	fruits = ["apple", "banana", "cherry"] if "banana" in fruits: print("yes")	yes
is	Pour vérifier si une valeur est présente dans une liste, un tuple, etc.	x = ["apple", "banana", "cherry"] y = x print(x is y)	True
lambda	Pour créer une fonction anonyme	x = lambda a : a + 10 print(x(5))	15
None	Représente une valeur nulle	x = None print(x)	None is not True, or False, None is just None...

nonlocal	Pour déclarer une variable non locale	def myfunc1():
		  x = "John"
		  def myfunc2():
		  nonlocal x
		  x = "hello"
		  myfunc2()
		  return x
		
		print(myfunc1())
			hello

not	Un opérateur logique ‘non’	x = False print(not x)	True

or	Un opérateur logique ‘ou’	if 5 > 3 or 5 > 10:
		  print("At least one of the statements are True")
		else:
		  print("None of the statements are True")
			At least one of the statements are True

pass	Une déclaration nulle, une déclaration qui ne fera rien	a = 33 b = 200 if b > a: pass # having an empty if statement like this, would raise an error without the pass statement	

raise	Pour lever une exception	x = -1
		
		if x < 0:
		  raise Exception("Sorry, no numbers below zero")
			Traceback (most recent call last):
			  File "demo_ref_keyword_raise.py", line 4, in <module>
			  raise Exception("Sorry, no numbers below zero")
			Exception: Sorry, no numbers below zero

return	Pour quitter une fonction et renvoyer une valeur	def myfunction(): return 3+3 print(myfunction())	6
True	Valeur booléenne, résultat des opérations de comparaison (vrai)	print(7 > 6)	True

try	Pour faire un essai... à l'exception de la déclaration	# (x > 3) will raise an error because x is not defined:
		
		try:
		  x > 3
		except:
		  print("Something went wrong")
		
		print("Even if it raised an error, the program keeps running")
			Something went wrong
			Even if it raised an error, the program keeps running

while	Pour créer une boucle while	x = 0
		
		while x < 9:
		  print(x)
		  x = x + 1
			0
			1
			2
			3
			4

			5 6 7 8
with	Utilisé pour simplifier la gestion des exceptions		
yield	Pour terminer une fonction, renvoie un générateur		

1/Création d'un commentaire

Les commentaires commencent par un #, et Python les ignorera :

#This is a comment.
print("Hello, World!")
	Hello, World!

- Ou, vous pouvez utiliser une chaîne multiligne entre les ”””

""" This is a comment written in more than just one line """ print("Hello, World!")	Hello, World!

2/Les Variables

Les variables sont des conteneurs pour stocker des valeurs de données.

Une variable est créée au moment où vous lui attribuez une valeur pour la première fois.

x = 5 y = "John" print(x) print(y)	5 John

Les variables n'ont pas besoin d'être déclarées avec un *type* particulier et peuvent même changer de type après avoir été définies.

x = 4 # x is of type int
x = "Sally" # x is now of type str
print(x)
	Sally

Si vous souhaitez spécifier le type de données d'une variable, vous pouvez le faire avec cast.

x = str(3) # x will be '3'
y = int(3) # y will be 3
z = float(3) # z will be 3.0
	3
	3
	3.0

Nomination des variables:

Une variable peut avoir un nom court (comme x et y) ou un nom plus descriptif (age, carname, total_volume). Règles pour les variables Python :

- Un nom de variable doit commencer par une lettre ou le caractère de soulignement
- Un nom de variable ne peut pas commencer par un chiffre
- Un nom de variable ne peut contenir que des caractères alphanumériques et des traits de soulignement (Az, 0-9 et _)

- Les noms de variables sont sensibles à la casse (age, Age et AGE sont trois variables différentes)
- Un nom de variable ne peut pas être l'un des mots-clés Python .

Exemples:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

Une valeur à plusieurs variables:

Vous pouvez attribuer la *même* valeur à plusieurs variables sur une seule ligne :

```
x = y = z = "Orange"
```

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
```

Variables de sortie: (print)

x = 5 y = "John" print(x, y)	5 John

Les Variables globales

Les variables créées en dehors d'une fonction sont appelées variables globales.

Les variables globales peuvent être utilisées par tout le monde, à la fois à l'intérieur des fonctions et à l'extérieur.

x = "awesome"
 	
def myfunc():
  x = "fantastic"
  print("Python is " + x)
 	
myfunc()
 	
print("Python is " + x)
	Python is fantastic
	Python is awesome

def myfunc():
  global x
  x = "fantastic"
 	
myfunc()
 	
print("Python is " + x)
	Python is fantastic

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`

Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

3/Types de données Python

Types de données intégrés:

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

- Vous pouvez obtenir le type de données de n'importe quel objet en utilisant la `type()`fonction :

x = 5 print(type(x))	<class 'int'>

Définition du type de données :

En Python, le type de données est défini lorsque vous affectez une valeur à une variable :

`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset

`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview
`x = None`	NoneType

Définition du type de données spécifique:

Si vous souhaitez spécifier le type de données, vous pouvez utiliser les fonctions constructeur suivantes :

`x = str("Hello World")`	str
`x = int(20)`	int
`x = float(20.5)`	float
`x = complex(1j)`	complex
`x = list(("apple", "banana", "cherry"))`	list
`x = tuple(("apple", "banana", "cherry"))`	tuple
`x = range(6)`	range
`x = dict(name="John", age=36)`	dict
`x = set(("apple", "banana", "cherry"))`	set
`x = frozenset(("apple", "banana", "cherry"))`	frozenset
`x = bool(5)`	bool
`x = bytes(5)`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

Nombres Python:

Il existe trois types numériques en Python :

- int
- float
- complex

Les variables de types numériques sont créées lorsque vous leur attribuez une valeur :

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

NB: Float peut également être un nombre scientifique avec un "e" pour indiquer la puissance de 10.

x = 35e3	35000.0
y = 12E4	120000.0
z = -87.7e100	-8.77e+101

Les nombres complexes s'écrivent avec un "j" comme partie imaginaire :

```
x = 3+5j
y = 5j
z = -5j
```

Conversion de types:

#convert from int to float:
x = float(1)
	1.0

#convert from float to int:
y = int(2.8)
	2

#convert from int to complex:
z = complex(1)
	(1+0j)

Nombre aléatoire:

Python n'a pas de `random()`fonction pour créer un nombre aléatoire, mais Python a un module intégré appelé `random` qui peut être utilisé pour créer des nombres aléatoires :

import random print(random.randrange(1, 10))	7

4/Python Strings :

Affecter une chaîne à une variable

a = "Hello" print(a)	Hello

Chaînes multilignes:

a = """Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.""" print(a)	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Obtenez un caractère à une position:

a = "Hello, World!" print(a[1])	e

Boucle sur une chaîne

Puisque les chaînes sont des tableaux, nous pouvons parcourir les caractères d'une chaîne, avec une boucle for.

for x in "banana": print(x)	b a n a n a

Longueur de chaine

a = "Hello, World!" print(len(a))	13

Vérifier la chaîne:

Pour vérifier si une certaine phrase ou un certain caractère est présent dans une chaîne, nous pouvons utiliser le mot-clé `in`.

txt = "The best things in life are free!" print("free" in txt)	True

Utilisez-le dans une déclaration `if`:

txt = "The best things in life are free!" if "free" in txt: print("Yes, 'free' is present.")	Yes, 'free' is present.

Pour vérifier si une certaine phrase ou un certain caractère n'est PAS présent dans une chaîne, nous pouvons utiliser le mot-clé `not in`.

txt = "The best things in life are free!" print("expensive" not in txt)	True

Utilisez-le dans une `if`déclaration :

txt = "The best things in life are free!" if "expensive" not in txt: print("No, 'expensive' is NOT present.")	No, 'expensive' is NOT present.

Tranchage:

Vous pouvez renvoyer une plage de caractères en utilisant la syntaxe slice.

Spécifiez l'index de début et l'index de fin, séparés par deux points, pour renvoyer une partie de la chaîne.

b = "Hello, World!" print(b[2:5])	llo

Trancher depuis le début

En omettant l'index de début, la plage commencera au premier caractère :

b = "Hello, World!" print(b[:5])	Hello

Trancher jusqu'au bout:

En omettant l' index *de fin* , la plage ira jusqu'à la fin :

b = "Hello, World!" print(b[2:])	llo, World!

Indexation négative:

Utilisez des index négatifs pour démarrer la tranche à partir de la fin de la chaîne :

b = "Hello, World!" print(b[-5:-2])	orl

Modifier les chaînes :

Majuscule : La `upper()`méthode renvoie la chaîne en majuscule :

a = "Hello, World!" print(a.upper())	**HELLO, WORLD!**

Minuscule : La `lower()`méthode renvoie la chaîne en minuscule :

a = "Hello, World!" print(a.lower())	hello, world!

Supprimer les espaces blancs : L'espace blanc est l'espace avant et/ou après le texte réel, et très souvent vous souhaitez supprimer cet espace.

La `strip()`méthode supprime tout espace blanc au début ou à la fin :

a = " Hello, World! " print(a.strip())	Hello, World!

Remplacer la chaîne : La `replace()`méthode remplace une chaîne par une autre :

a = "Hello, World!" print(a.replace("H", "J"))	Jello, World!

Séparer la chaîne : La `split()`méthode renvoie des sous-chaines dans une liste dans laquelle le texte entre le séparateur spécifié devient les éléments de la liste.

a = "Hello, World!" b = a.split(",") print(b)	['Hello', ' World!']

Concaténation de chaînes : Pour concaténer ou combiner deux chaînes, vous pouvez utiliser l'opérateur +.

Fusionner variable `a`avec variable `b` dans variable `c` :

a = "Hello" b = "World" c = a + b print(c)	HelloWorld

Pour ajouter un espace entre eux, ajoutez un `" "`:

a = "Hello" b = "World" c = a + " " + b print(c)	Hello World

Format de chaîne : La `format()`méthode prend les arguments passés, les formate et les place dans la chaîne où `{}` se trouvent les espaces réservés :

age = 36 txt = "My name is John, and I am {}" print(txt.format(age))	My name is John, and I am 36
quantity = 3 itemno = 567 price = 49.95 myorder = "I want {} pieces of item {} for {} dollars." print(myorder.format(quantity, itemno, price))	I want 3 pieces of item 567 for 49.95 dollars.

quantity = 3 itemno = 567 price = 49.95 myorder = "I want to pay {2} dollars for {0} pieces of item {1}." print(myorder.format(quantity, itemno, price))	I want to pay 49.95 dollars for 3 pieces of item 567

Caractères d'échappement : Pour insérer des caractères non autorisés dans une chaîne, utilisez un caractère d'échappement.

Un caractère d'échappement est une barre oblique inverse \ suivie du caractère que vous souhaitez insérer.

\'	Single Quote	txt = 'It\'s alright.' print(txt)	It's alright.
\\	Backslash	txt = "This will insert one \\ (backslash)." print(txt)	This will insert one \ (backslash).
\n	New Line	txt = "Hello\nWorld!" print(txt)	Hello World!
\r	Carriage Return	txt = "Hello\rWorld!" print(txt)	Hello World!
\t	Tab	txt = "Hello\tWorld!" print(txt)	Hello World!

\b	Backspace	#This example erases one character (backspace):
		txt = "Hello \bWorld!"
		print(txt)
			HelloWorld!

\f	Form Feed	Saut de formulaire	

\ooo	Octal value	#A backslash followed by three integers will result in a octal value:
		txt = "\110\145\154\154\157"
		print(txt)
			Hello

\xhh	Hex value	#A backslash followed by an 'x' and a hex number represents a hex value:
		txt = "\x48\x65\x6c\x6c\x6f"
		print(txt)
			Hello

Méthodes de chaîne :

Python possède un ensemble de méthodes intégrées que vous pouvez utiliser sur des chaînes.

Remarque : Toutes les méthodes de chaîne renvoient de nouvelles valeurs. Ils ne changent pas la chaîne d'origine.

Method	Description
capitalize()	Convertit le premier caractère en majuscule

casefold()	Convertit la chaîne en minuscules
center()	Renvoie une chaîne centrée
count()	Renvoie le nombre de fois qu'une valeur spécifiée apparaît dans une chaîne
encode()	Renvoie une version codée de la chaîne
endswith()	Renvoie vrai si la chaîne se termine par la valeur spécifiée
expandtabs()	Définit la taille de tabulation de la chaîne
find()	Recherche la chaîne pour une valeur spécifiée et renvoie la position de l'endroit où elle a été trouvée
format()	Formate les valeurs spécifiées dans une chaîne
format_map()	Formate les valeurs spécifiées dans une chaîne
index()	Recherche la chaîne pour une valeur spécifiée et renvoie la position de l'endroit où elle a été trouvée
isalnum()	Renvoie True si tous les caractères de la chaîne sont alphanumériques
isalpha()	Renvoie True si tous les caractères de la chaîne sont dans l'alphabet
isdecimal()	Renvoie True si tous les caractères de la chaîne sont des décimales
isdigit()	Renvoie True si tous les caractères de la chaîne sont des chiffres
isidentifier()	Renvoie True si la chaîne est un identifiant
islower()	Renvoie True si tous les caractères de la chaîne sont en minuscules
isnumeric()	Renvoie True si tous les caractères de la chaîne sont numériques
isprintable()	Renvoie True si tous les caractères de la chaîne sont imprimables
isspace()	Renvoie True si tous les caractères de la chaîne sont des espaces
istitle()	Renvoie True si la chaîne suit les règles d'un titre
isupper()	Renvoie True si tous les caractères de la chaîne sont en majuscules
join()	Joint les éléments d'un itérable à la fin de la chaîne
ljust()	Renvoie une version justifiée à gauche de la chaîne
lower()	Convertit une chaîne en minuscule
lstrip()	Renvoie une version trim gauche de la chaîne
maketrans()	Renvoie une table de traduction à utiliser dans les traductions
partition()	Renvoie un tuple où la chaîne est divisée en trois parties

replace()	Renvoie une chaîne où une valeur spécifiée est remplacée par une valeur spécifiée
rfind()	Recherche la chaîne pour une valeur spécifiée et renvoie la dernière position où elle a été trouvée
rindex()	Recherche la chaîne pour une valeur spécifiée et renvoie la dernière position où elle a été trouvée
rjust()	Renvoie une version justifiée à droite de la chaîne
rpartition()	Renvoie un tuple où la chaîne est divisée en trois parties
rsplit()	Fractionne la chaîne au niveau du séparateur spécifié et renvoie une liste
rstrip()	Renvoie une version ajustée à droite de la chaîne
split()	Fractionne la chaîne au niveau du séparateur spécifié et renvoie une liste
splitlines()	Fractionne la chaîne aux sauts de ligne et renvoie une liste
startswith()	Renvoie vrai si la chaîne commence par la valeur spécifiée
strip()	Renvoie une version tronquée de la chaîne
swapcase()	Échange de casse, les minuscules deviennent des majuscules et vice versa
title()	Convertit le premier caractère de chaque mot en majuscule
translate()	Renvoie une chaîne traduite
upper()	Convertit une chaîne en majuscule
zfill()	Remplit la chaîne avec un nombre spécifié de valeurs 0 au début

5/Valeurs booléennes :

Vous pouvez évaluer n'importe quelle expression en Python et obtenir l'une des deux réponses, `True`ou `False`.

Lorsque vous comparez deux valeurs, l'expression est évaluée et Python renvoie la réponse booléenne :

print(10 > 9) **print(10 == 9)** **print(10 < 9)**	**True** **False** **False**

Imprimer un message selon que la condition est `True` ou `False` :

a = 200
b = 33
 	
if b > a:
  print("b is greater than a")
else:
	b is not greater than a

print("b is not greater than a")	

La `bool()`fonction vous permet d'évaluer n'importe quelle valeur, et de vous donner `True` ou `False` en retour,

print(bool("Hello")) print(bool(15))	True True

NB:

Toute chaîne est `True`, sauf les chaînes vides.

Tout nombre est `True`, sauf `0`.

Toutes les listes, tuples, ensembles et dictionnaires sont `True`, sauf ceux vides.

NB:

En fait, il n'y a pas beaucoup de valeurs évaluées à `False`, à l'exception des valeurs vides, telles que `()`, `[]`, `{}`, `""`, le nombre `0`et la valeur `None`. Et bien sûr, la valeur `False` est évaluée à `False`. un objet créé à partir d'une classe avec une `__len__`fonction qui renvoie `0` or `False`

Aussi: Vous pouvez créer des fonctions qui renvoient une valeur booléenne :

def myFunction() : return True print(myFunction())	True

la `isinstance()` fonction, qui peut être utilisée pour déterminer si un objet est d'un certain type de données :

x = 200 print(isinstance(x, int))	True