Présentation du langage Python 3

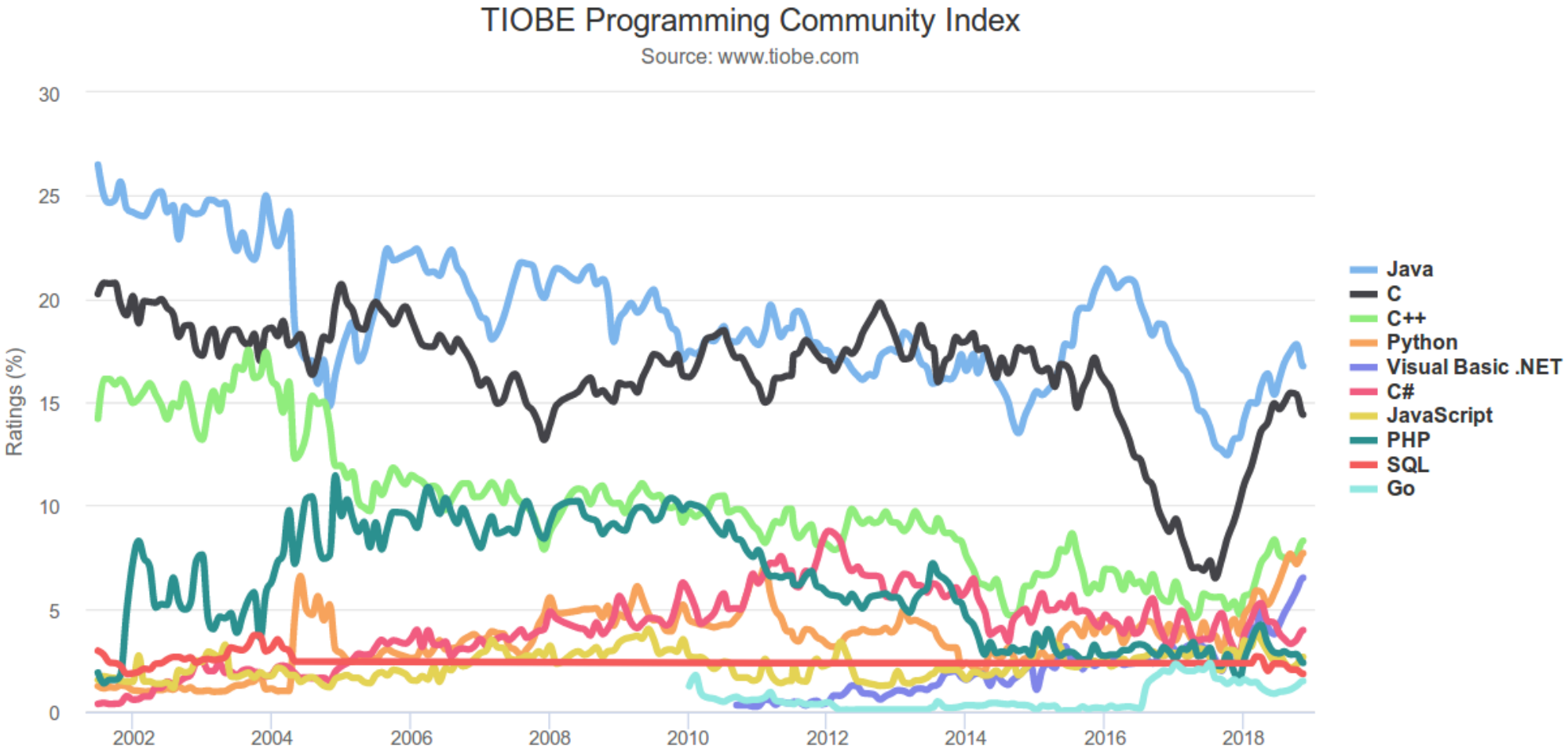
Ce diaporama s’adresse aux débutants en Python ayant déjà une expérience de la programmation.

En première lecture, vous pourrez ignorer les diapositives qui commencent par une étoile (*).

Présentation du langage Python 3

- Création en 1989 par Guido van Rossum « BDFL »
- Licence libre (GPL)
- Langage interprété, orienté objet
- Multiplateforme (Windows, Linux, Mac, Android, etc.)
- Facile à apprendre
- Populaire dans le milieu éducatif et universitaire
- Usage professionnel

Classement TIOBE



Python est l'un des langages généralistes les plus populaires, avec Java, C ou C++

Versions

- La version majeure en cours est la version 3 mais la version 2 est encore largement utilisée.

- Pas de compatibilité ascendante entre la version 2 et la version 3 :(

Installation

- Par la suite, on s'intéressera uniquement à l'implémentation standard de Python (CPython).

- Téléchargement et procédure d'installation :

  Python 3.8.2 (février 2020)

  Python 2.7.17 (octobre 2019)

  https://www.python.org/downloads/

Modules (librairies)

- Le richesse de Python réside dans le nombre impressionnant de modules disponibles (200 000 actuellement)
- « Package » = module de modules

- PyPI - the Python Package Index

  https://pypi.python.org/pypi

Quelques modules célèbres

- calcul scientifique (*NumPy* et *SciPy*)
- graphiques (*matplotlib*)
- traitement d'images (*PIL*)
- vision artificielle par caméra (framework *SimpleCV*)
- data mining, machine learning, deep learning (*scikit-learn, TensorFlow, Keras*)
- bio-informatique (*Biopython*)
- interface graphique (*Tkinter*, *PyQt*, *wxPython*, *PyGTK*)
- applications Web (serveur Web *Zope* ; frameworks Web *Flask*, *Django*)

Quelques modules célèbres

- systèmes de gestion de base de données (*SQLalchemy*)
- analyse big data (*pandas*)
- applications réseau (*socket*, framework *twisted*)
- communication avec port série (*PySerial*), Bluetooth (*pybluez*)
- multitâches (*threading*)
- programmation asynchrone (*asyncio*)

- les modules *py2exe* (sous Windows) et *cx_Freeze* permettent de rendre vos scripts Python exécutables :)

Modules « built-in », standards et autres

- Les modules built-in sont les modules natifs de Python.

  Ils sont toujours disponibles :

  - *math*, *time*, *sys,* …

- Les modules standards sont des modules utiles que l'on a intégré dans la distribution Python :

  - *socket, urllib, threading, sqlite3, ...*

Modules « built-in », standards et autres

- Les modules externes doivent être ajoutés, la procédure d'installation est variable suivant l'OS :
  - Exemple : pour installer le module *matplotlib* sur une Raspberry Pi (distribution Linux Raspbian), il suffit de taper la commande système suivante :

    sudo apt install python-matplotlib (python 2)

    sudo apt install python3-matplotlib (python 3)
  - Autrement, l'utilitaire **pip** doit marcher dans la plupart des cas :

    python -m pip install matplotlib

Python et C/C++ sont complémentaires

- Intégration dans Python de modules écrits en C/C++

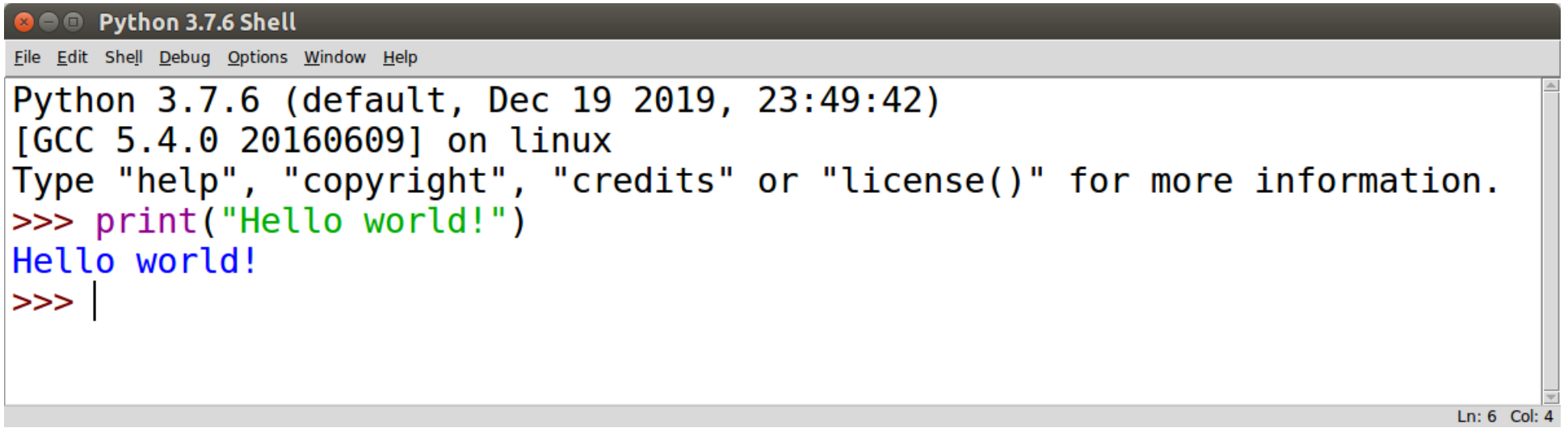
  => gain en performance

  Les modules built-in sont écrits en C.

- Intégration dans C/C++ de modules écrits en Python

Environnement de développement (IDE)

- IDE par défaut : IDLE
  - Éditeur de code
  - Interpréteur
  - Débogueur



- Dans Windows : Démarrer → Rechercher : IDLE

Environnement de développement (IDE)

- Autres IDE
  - Eclipse (avec le plugin pydev)
  - Eric
  - Spyder
  - Thonny
  - ipython (en mode console)
  - Jupyter notebook
  - Geany
  - Notepad++

Python 3 – Les variables

- Typage dynamique (déclaration d'une variable sans préciser explicitement son type)
- Principaux types :
  - Entier (int)
  - Nombre « flottant » (float)
  - Chaîne de caractères (str)
  - Booléen (bool)
  - Liste (list)
  - Tuple (tuple)
  - Dictionnaire (dict)

Python 3 – Type int

```
>>> a = 5
>>> print(a)
5
```

- Dans l'interpréteur, la fonction print() est facultative :

```
>>> a
5
>>> print(type(a))   # ou type(a)
<class 'int'>
>>> b, c = 9, 3   # affectation de plusieurs variables
>>> b - c      # ou print(b - c)
6
```

Python 3 – Type float

```
>>> n = 3    # type int
>>> n = n + 0.2   # donne un type float
>>> n
3.2
>>> a = 12.0    # type float
>>> b = -8.23e3
>>> c = 81.5385
>>> D = b**2 - 4*a*c
>>> D
67728986.152
```

Python 3 – Dernier résultat

- Dans un interpréteur, le dernier résultat d'un calcul est disponible dans la variable « _ » :

```
>>> 10 + 2
12
>>> _
12
>>> _ * 5
60
>>> _
60
```

Python 3 – Le module math

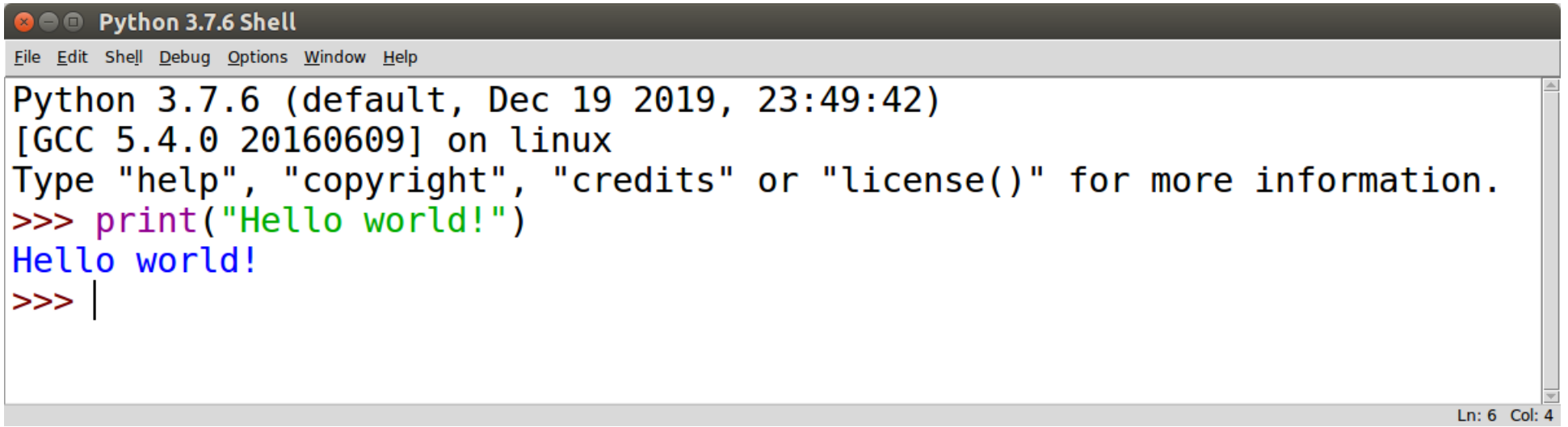
```
>>> import math
>>> a = math.sqrt(3)/2
>>> a
0.8660254037844386
>>> b = math.sin(math.pi/3)
>>> b
0.8660254037844386
>>> math.log10(1e6)
6.0
```

Python 3 – La fonction help()

```
>>> help(math)   # aide sur le module math
>>> help(math.sin)
Help on built-in function sin in module math:
sin(...)
    sin(x)
    Return the sine of x (measured in radians).
```

Python 3 – Type str (chaîne de caractères)

Le « Hello world ! » en Python demande une seule ligne de code :



Python 3 – Type str (chaîne de caractères)

```
>>> nom = 'Dupont'
>>> nom
'Dupont'
>>> print(nom)    # n'affiche pas les délimiteurs
Dupont
>>> prenom = 'Pierre'
```

- Concaténation

```
>>> chaine = prenom + '  ' + nom
>>> chaine
'Pierre  Dupont'
```

Python 3 – Type str (chaîne de caractères)

- Saut de ligne avec la séquence d'échappement « \n »

```
>>> chaine ='Première ligne\nDeuxième ligne\nTroisième ligne'
```

- ou plus simplement :

```
>>> chaine =  " " "Première ligne
Deuxième ligne
Troisième ligne" " "
>>> print(chaine)
Première ligne
Deuxième ligne
Troisième ligne
```

*Python 3 – Type str (chaîne de caractères)

- Autres séquences d’échappement :

```
>>> chaine = '\\ \' \"\n\x61'
>>> print(chaine)
\ ' "
a
```

- Les « raws strings »

```
>>> chaine = r'\\ \' \"\n\x61'
>>> print(chaine)
\\ \' \"\n\x61
```

Python 3 – La fonction input()

- En mode console
- Retourne un type str

```
>>> nom = input("Entrer votre nom : ")
Entrer votre nom : Dupont
>>> nom
'Dupont'
```

Python 3 – Conversion de types

- Fonctions int(), float() et str()

```
>>> nombre = float(input("Entrer un nombre : "))
Entrer un nombre : 4
>>> nombre**2
16.0
```

Python 3 – Formatage des données

- Fonction format()

```
>>> nom, age, masse = 'Dupont', 18, 72.4
>>> print("Mon nom est {}, âge {} ans et masse {} kg".format(nom, age, masse))
Mon nom est Dupont, âge 18 ans et masse 72.4 kg
>>> print("Mon nom est {}, âge {} ans et masse {:.3f} kg".format(nom, age, masse))
Mon nom est Dupont, âge 18 ans et masse 72.400 kg
```

Python 3 – Formatage des données

- Pour python >= 3.6 : les « f-strings »

```
>>> nom, age, masse = 'Dupont', 18, 72.4
>>> print(f"Mon nom est {nom}, âge {age} ans et masse {masse} kg")
Mon nom est Dupont, âge 18 ans et masse 72.4 kg
```

Python 3 – Le type list

- Une liste est une structure de données, une sorte de tableau.
- Le premier élément (item) d'une liste possède l'indice (l'index) 0.
- Dans une liste, on peut avoir des éléments de types différents.

```
>>> infoperso = ['Pierre', 'Dupont', 18, 1.75]
>>> infoperso[2]          # le troisième élément
18
```

Python 3 – Le type list

- On peut modifier les éléments d'une liste :

  >>> infoperso[2] = 19

  >>> infoperso

  ['Pierre', 'Dupont', 19, 1.75]

- La taille d'une liste peut aussi être modifiée.

  Par exemple, on peut ajouter un élément en fin de liste avec la méthode append() :

  >>> infoperso.append(72.4)

  >>> infoperso

  ['Pierre', 'Dupont', 19, 1.75, 72.4]

*Python 3 – Le type list

- Un exemple avec une liste d'attente (dans le genre pile FIFO) :

  >>> L = [12, 15, 20, 22, 28, 34, 43, 59]

  On supprime le dernier élément (59) :

  >>> L.pop()

  On insère un élément (10) en début de liste (indice 0) :

  >>> L.insert(0, 10)

  >>> L

  [10, 12, 15, 20, 22, 28, 34, 43]

- Vous noterez que nous n'avons pas eu besoin de faire une boucle pour décaler les éléments.

Python 3 – La fonction range()

- range() permet de créer une séquence d'entiers :

```
>>> sequence = range(5)
>>> sequence    # objet de type « range »
range(0, 5)
>>> list(sequence)   # conversion en liste pour plus de clarté
[0, 1, 2, 3, 4]
>>> sequence2 = range(1, 11)   # début, fin non comprise
>>> list(sequence2)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(4, -1, -1))
[4, 3, 2, 1, 0]
```

Python 3 – Le type tuple

- Un tuple ressemble beaucoup à une liste :

```
>>> infoperso = ('Pierre', 'Dupont', 18, 1.75)
>>> infoperso = 'Pierre', 'Dupont', 18, 1.75    # autre écriture
>>> infoperso[2]
18
```

- Mais contrairement à une liste, on ne peut pas modifier un tuple (un tuple est non « mutable ») :

```
>>> infoperso[2] = 19
TypeError: 'tuple' object does not support item assignment
```

Python 3 – Le slicing

- Le slicing (« tranchage ») concerne les objets indexables : chaîne de caractères, liste, tuple.

```
>>> nom = 'Durand'
>>> nom[1:4]     # éléments d'indices 1 à 3
'ura'
>>> nom[4:]
'nd'
>>> nom[-2]     # avant dernier élément
'n'
>>> nom[::-1]   # -1 désigne le pas
'dnaruD'
```

Python 3 – Permuter des variables

- En Python, on peut permuter deux variables (ou plus) sans passer par une variable intermédiaire :

```
>>> a = 4
>>> b = 8
>>> b, a = a, b
>>> a
8
>>> b
4
```

Python 3 – Le type dict

- Un dictionnaire stocke des données sous la forme clé ⇒ valeur (c'est une sorte de tableau associatif).
- Une clé est unique et n'est pas nécessairement un entier (comme c'est le cas de l'indice d'une liste ou d'un tuple).

```
>>> moyennes = {'maths': 12.5, 'anglais': 15.8}      # entre accolades
>>> moyennes['anglais']          # entre crochets
15.8
>>> moyennes['anglais'] = 14.3          # nouvelle affectation
>>> moyennes['sport'] = 11.0                # nouvelle entrée
>>> moyennes
{'sport': 11.0, 'anglais': 14.3, 'maths': 12.5}
```

*Python 3 – Le type dict

- Remarques : pas de relation d'ordre pour les clés.

  Les clés sont généralement des chaînes de caractères, mais aussi des nombres ou des tuples.

  Attention, les listes ne sont pas autorisées.

  Un exemple avec des clés de type tuple :

```
>>> pixels = {(0, 0): 'blue', (0, 1): 'red', (1, 0): 'white', (1, 1): 'red'}
>>> pixels[0, 1]
'red'
>>> pixels[0, 1] = 'black'
>>> pixels[2, 0] = 'green'        # un nouveau pixel
```

Python 3 – Le type bool (booléen)

- Deux valeurs sont possibles : True et False
- Opérateurs de comparaison :

  <

  <=

  >

  >=

  ==

  !=

Python 3 – Le type bool (booléen)

```
>>> b = 10
>>> b > 8
True
>>> b == 5
False
>>> 0 <= b <= 10    # une écriture bien pratique
True
```

Python 3 – Le type bool (booléen)

- Les opérateurs logiques : and, or, not

```
>>> note = 13.5
>>> horslimite = not(0.0 <= note <= 20.0)
>>> horslimite
False
>>> age, taille = 13, 155
>>> age > 12 and taille > 140
True
```

*Python 3 – Le type bool (booléen)

- La classe bool hérite de la classe int, c'est-à-dire qu'elle reprend les méthodes de la classe int

  False correspond alors à 0 et True à 1 :

```
>>> 1+True    # héritage de l'addition de la classe int
2
>>> 5*(3>1)
5
>>> False == 0
True
```

Python 3 – L'opérateur in

- L'opérateur in s'utilise avec les chaînes (type str), les listes, les tuples ainsi que les dictionnaires :

```
>>> chaine = 'Bonsoir'
>>> 'soir' in chaine
True
>>> maliste = [4, 8, 15]
>>> 9 in maliste
False
```

Python 3 – Les conditions

- Syntaxe

```
if expression :
        bloc d'instructions    # indentation
else :              # else doit être au même niveau que l'instruction if
        bloc d'instructions    # indentation
# suite du programme
```

- L'indentation est libre mais l'usage est de prendre 4 espaces (à paramétrer dans votre IDE favori).
- La délimitation d'un bloc de code par indentation fait qu'un programme Python est naturellement lisible (mais pas forcement bien écrit...).

Python 3 – Les conditions

```
temperature = float(input("Température en °C : "))
if temperature >= 19.0:
        print("Il fait bon")
else:
        print("Il fait froid")
print("Suite du programme")
```

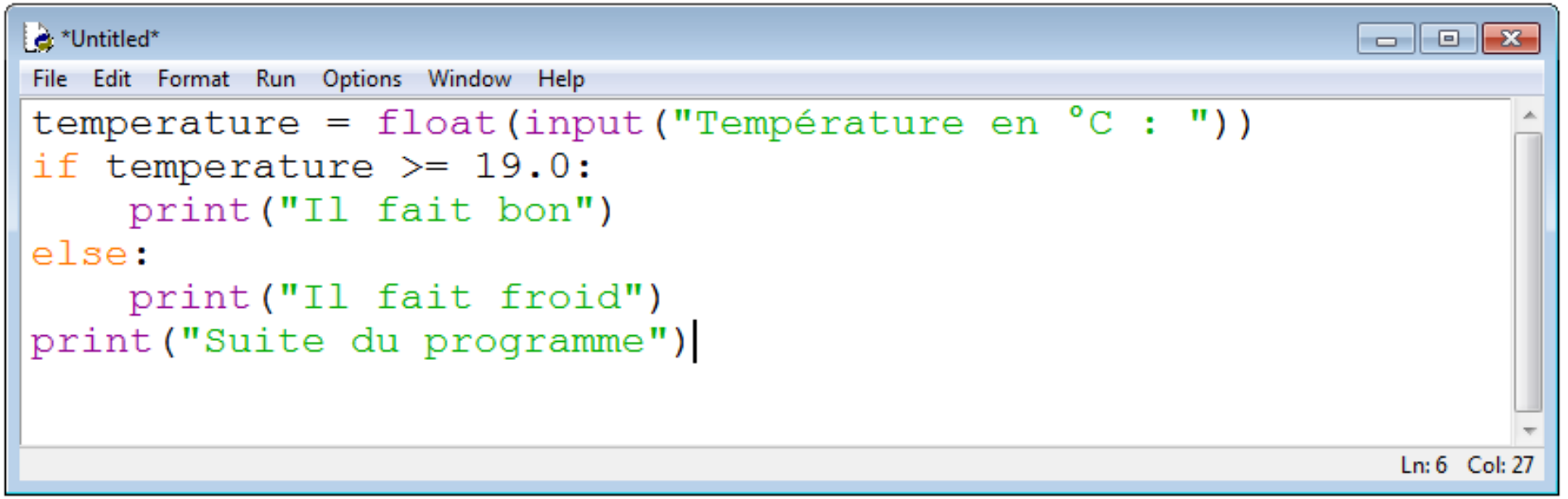
```
>>> Température en °C : 20
Il fait bon
Suite du programme
>>>
```

Python 3 – Premier script

- Script = programme en Python
- Dans Windows : Démarrer → Rechercher : IDLE

  File → New File

  Vous pouvez maintenant saisir le code de votre script :

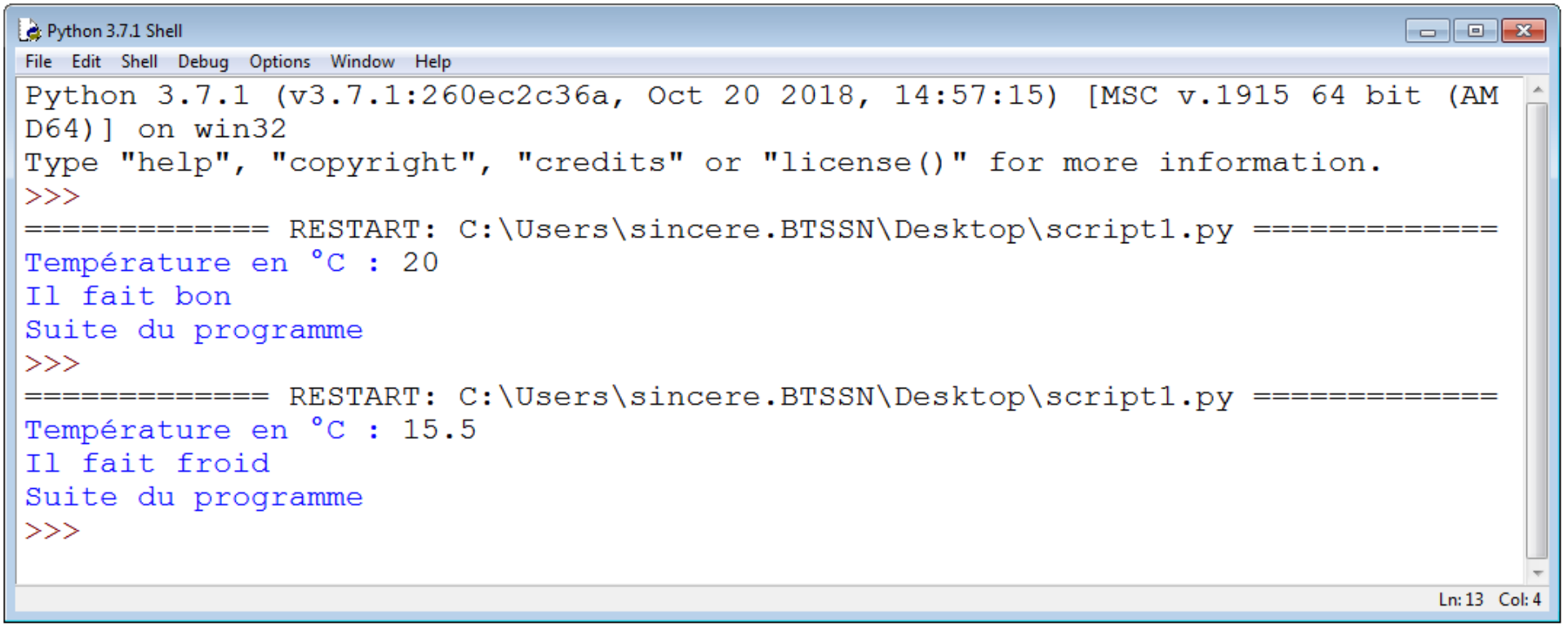


Python 3 – Premier script

- File → Save as

  Nom du fichier : script1.py (avec l'extension .py)

- Run (ou touche F5)



Python 3 – Les conditions

- Instruction elif (contraction de else if : sinon si)

```
temperature = float(input("Température en °C : "))
if temperature >= 36.0:
        print("Canicule !")
elif temperature >= 19.0 :
        print("Il fait bon")
else:
        print("Il fait froid")
print("Suite du programme")
```

Python 3 – Les conditions

- Remarque : pas de switch / case / break comme en langage C
- Cela a été jugé inutile en Python car le code suivant fait la même chose, en plus concis et avec plus de souplesse :

```
cas = 2
if cas == 1:
        print("Cas 1")
elif cas == 2:
        print("Cas 2")
elif 3 <= cas <= 9:
        print("Cas 3 à 9")
else :
        print("Autre cas")
```

*Python 3 – Les expressions conditionnelles

- C’est un raccourci d’écriture pour initialiser une variable suivant une condition :

```
>>> b = 5
>>> a = 20 if b > 10 else 0
>>> a
0
>>> b = 15
>>> a = 20 if b > 10 else 0
>>> a
20
```

Python 3 – Les boucles while

- Syntaxe

```
while expression :
        bloc d'instructions          # indentation
# suite du programme
```

Python 3 – Les boucles while

```
i = 1      # initialisation de la variable de comptage
while i < 5:
        print(i)
        i += 1    # incrémentation  i = i + 1
# suite du programme


>>>
1
2
3
4
```

Python 3 – Les boucles while

- L'instruction break

```
i = 1
while True:
        print(i)
        i += 1    # incrémentation
        if i > 4 :
                break
>>>
1
2
3
4
```

Python 3 – Les boucles while

- L'instruction continue

```
i = 0
while i < 5 :
        i += 1
        if i == 3 :
                continue
        print(i)
>>>
1
2
4
5
```

Python 3 – Les boucles while

- Remarque : pas de do / while comme en langage C
- Le code suivant permet de faire la même chose :

```
while True:
        bloc d'instructions    # exécuté au moins une fois
        if expression:
                break
# suite du programme
```

Python 3 – Les boucles for

- Syntaxe

```
for element in sequence :
        bloc d'instructions     # indentation
# suite du programme
```

- Les éléments (items) de la séquence sont issus d'un objet « itérable » :

  chaîne de caractères, liste, tuple, dictionnaire ou range.

Python 3 – Les boucles for

```
for i in range(5):
        print(i)
# suite du programme
>>>
0
1
2
3
4
```

Python 3 – Les boucles for

- La manière pythonique de parcourir une liste est la suivante :

```
infoperso = ['Pierre', 'Dupont', 18, 1.75]    # une liste
for i in infoperso:
        print(i)
# suite du programme
>>>
Pierre
Dupont
18
1.75
```

*Python 3 – Les boucles for

- Notez que cela remplace avantageusement la façon classique de parcourir une séquence :

```
# code classique à proscrire
infoperso = ['Pierre', 'Dupont', 18, 1.75]
for i in range(len(infoperso)):
        print(infoperso[i])
>>>
Pierre
Dupont
18
1.75
```

*Python 3 – Les boucles for

- Pour parcourir un dictionnaire :

```
moyennes = {'maths': 12.5, 'anglais': 15.8, 'sport': 11.0}
for cle, valeur in moyennes.items():
        print(cle, valeur)
```

```
>>>
sport 11.0
anglais 15.8
maths 12.5
```

*Python 3 – for … else

```
notes = [15, "Absent", 8, 122, "Absent", 14]    # une liste
for i in notes:
        if i == "Absent":
                continue     # on ignore les absences
        if not(0 <= i <= 20):
                print("Erreur : Note invalide !")
                break          # on sort de la boucle
        print(i)    # on affiche les notes valides
else:
        # ce bloc est exécuté si et seulement si la boucle for s'est déroulée normalement (pas de break)
        print("OK")
print("Suite du programme")
```

*Python 3 – for … else

```
>>>
15
8
Erreur : note invalide !
Suite du programme
```

- Avec : notes = [15, "Absent", 8, 12, "Absent", 14], on obtient :

```
15
8
12
14
OK
Suite du programme
```

- Remarque : on a également la structure while … else

*Python 3 – La fonction enumerate()

- Cette fonction s'utilise avec les chaînes de caractères, les listes, les tuples. Elle permet d'itérer conjointement sur l'indice et l'élément :

```
chaine = 'Python'
for indice, lettre in enumerate(chaine):
        print(indice, lettre)
>>>
0 P
1 y
2 t
3 h
4 o
5 n
```

*Python 3 – La fonction enumerate()

```
maliste = [12, 18, 21, 29, 33]
for indice, valeur in enumerate(maliste):
        print(indice, valeur)
>>>
0 12
1 18
2 21
3 29
4 33
```

*Python 3 – Les compréhensions

- Il s'agit d'un raccourci d'écriture pratique et puissant.

Soit à créer la liste des carrés des 100 premiers entiers.

Le code classique est le suivant :

```
carres = [ ]     # création d'une liste vide
for i in range(1, 101):
        carres.append(i**2)
>>> carres
[1, 4, 9, 16, 25, 36, 49, 64, 81, … , 10000]
```

*Python 3 – Les compréhensions

- La même chose avec une compréhension de liste, en une seule ligne de code :

```
>>> carres = [ i**2 for i in range(1, 101)]
>>> carres
[1, 4, 9, 16, 25, 36, 49, 64, 81, … , 10000]
```

- On peut même ajouter une condition :

```
>>> carres_centaines = [ i**2 for i in range(1, 101) if 100 <= i**2 < 1000]
>>> carres_centaines
[100, 121, 144, 169, 196, 225, 256, 289, …, 961]
```

*Python 3 – Les compréhensions

- Un exemple avec une compréhension de dictionnaire :

```
>>> moyennes = { i : float(input(i+" ? ")) for i in ['maths', 'anglais', 'sport'] }
maths ? 12
anglais ? 14.5
sport ? 11
>>> moyennes
{'sport': 11.0, 'maths': 12.0, 'anglais': 14.5}
```

Python 3 – Les fonctions

```
def nomdelafonction(parametre1, parametre2, etc.):
        " " " Documentation
qu'on peut écrire
sur plusieurs lignes " " "          # docstring
        bloc d'instructions          # indentation
        return resultat          # la valeur de retour de la fonction
```

Python 3 – Les fonctions

```
def fahrenheit(degrecelsius):
    "Conversion degré Celsius en degré Fahrenheit"
    resultat = degrecelsius*9/5 + 32
    return resultat


>>> fahrenheit(100)    # appel de la fonction avec l'argument 100
212.0
>>> help(fahrenheit)
Help on function fahrenheit :
fahrenheit(degrecelsius)
    Conversion degré Celsius en degré Fahrenheit
```

Python 3 – Les fonctions

- Une fonction avec un paramètre optionnel :

```
def ecart(n, ref=10):
        return n - ref


>>> ecart(12)
2
>>> ecart(31, 20)
11
>>> ecart(ref=5, n=4)
-1
```

*Python 3 – Les fonctions

- Une fonction avec un nombre arbitraire de paramètres :

```
def somme(*valeurs):    # avec une étoile * devant le nom
        result = 0
        for i in valeurs:      # valeurs est vu comme un tuple
                result += i
        return result
>>> somme(5)
5
>>> somme(8, 10, 5, 9)
32
```

*Python 3 – Les fonctions

- Une fonction avec un nombre arbitraire de paramètres nommés :

```
def bulletin(**matieres):   # avec deux étoiles ** devant le nom
        # matieres est vu comme un dictionnaire
        for matiere, note in matieres.items():
                if note >= 16:
                        print("Très bien en {}".format(matiere))
        return matieres
>>> notes = bulletin(maths=15, sport=10, anglais=18)
Très bien en anglais
>>> notes
{'sport': 10, 'maths': 15, 'anglais': 18}
```

Python 3 – La constante « None »

- Il est courant qu'une fonction ne retourne « rien » :

```
def affichage(message):
        print(message)
        return      # facultatif
```

- En vérité, une fonction retourne toujours quelque chose, ici l'objet None :

```
>>> resultat = affichage('Bonsoir')
Bonsoir
>>> print(resultat)
None
```

- Remarque : si vous voulez créer une variable qui n'a pas (encore) de valeur, vous pouvez l'initialiser avec None :

```
date_fin_du_monde = None
```

Python 3 – Variables globales et locales

```
a, b  = 10, 5        # variables globales
def mafonction():
        global a     # on fait maintenant référence à la variable globale
        a = 20       # variable globale
        b = 100      # création d'une variable locale
        return a, b      # on retourne un tuple
>>> mafonction()
(20, 100)
>>> a, b
(20, 5)
```

Python 3 – L'instruction pass

- L’instruction pass ne fait rien mais elle ne sert pas à rien :

  pass est syntaxiquement nécessaire pour remplir un bloc vide (un commentaire ne suffit pas en Python) :

```
def mafonction():
        pass

if condition is False:
        pass
```

*Python 3 – Le dépaquetage « unpacking » de séquence

- Il s'agit d'une syntaxe qui s'applique aux objets itérables : chaîne de caractères, liste, tuple, dictionnaire ou range.

```
>>> a, b, c = [3, 1 , 4]
>>> print(a, b, c)
3     1     4
```

- Un « dépaquetage » plus sophistiqué (notez bien la présence d'une étoile *) :

```
>>> notes = [8, 15, 17, 9, 12]
>>> premierenote, *autresnotes, dernierenote = notes
>>> print(premierenote, dernierenote, autresnotes)
8     12     [15, 17, 9]
```

A comparer avec un code classique :

```
>>> premierenote, dernierenote, autresnotes = notes[0], notes[-1], notes[1:-1]
```

*Python 3 – Le dépaquetage « unpacking » de séquence et les arguments de fonctions

```
def addition(x, y):
        return x + y
>>> t = 2, 5    # avec un tuple, ou une liste [2, 5]
>>> addition(t)     # donne une erreur
TypeError: addition() missing 1 required positional argument: 'y'
>>> addition(*t)     # avec une étoile *
7
```

- Cela fonctionne aussi avec un dictionnaire :

```
>>> d = {'x': 4, 'y': 2.5}
>>> addition(**d)     # avec deux étoiles **
6.5
```

Python 3 – Exceptions

- Exécutons ce script :

```
valeur = 0
inverse = 1 / valeur
print(inverse)
print("Suite du programme")
```

- Une exception est « levée », le programme s'arrête sur ce message d'erreur :

```
>>>
ZeroDivisionError: division by zero
```

Python 3 – Gestion des exceptions

- Le même script avec une gestion des exceptions, c'est-à-dire un traitement qui évite l'arrêt du programme :

```
valeur = 0
try :
        inverse = 1 / valeur
        print(inverse)
except ZeroDivisionError:    # traitement de l'exception
        print("Oups ! Division par zéro !")
print("Suite du programme")
```

```
>>> Oups ! Division par zéro !
Suite du programme
```

Python 3 – Gestion des exceptions

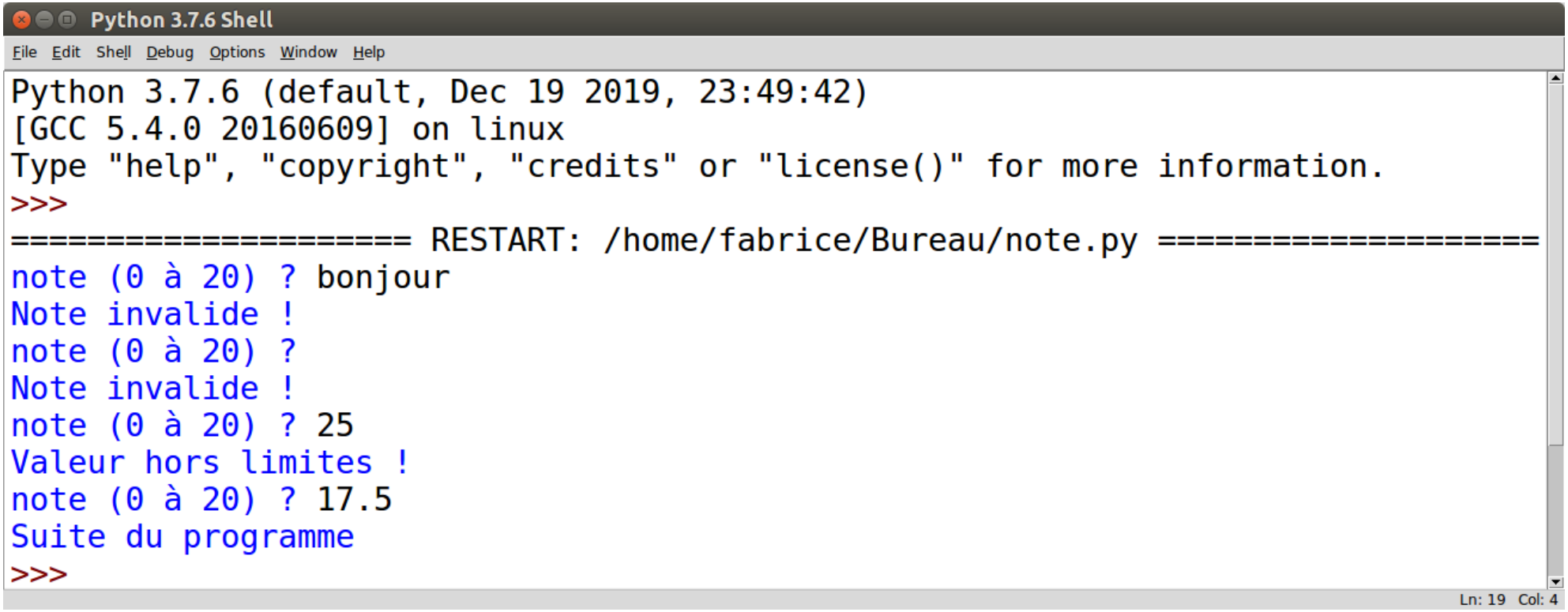
- Un autre exemple qui demande un nombre entre 0 et 20, avec un contrôle du format :

File Edit Format Run Options Window Help

```
# -*- coding: utf8 -*-
while True:
    try:
        note = float(input("note (0 à 20) ? "))
        if 0 <= note <= 20:
            break
        print("Valeur hors limites !")
    except ValueError:
        print("Note invalide !")
print("Suite du programme")
```

Ln: 11 Col: 0

Python 3 – Gestion des exceptions



Python 3 – Sortir d'un programme

- Fonction exit()

```
try :
        import matplotlib
except ImportError:
        print("Impossible d'importer le module matplotlib")
        exit(1)   # on quitte le programme avec un code d'erreur
# on continue ici si l'importation a réussi
```

Python 3 – Programmation orientée objet

« Everything in Python is an object »

- Comme en C++, on retrouve les notions :
  - de classe
  - d'instances de classe
  - les méthodes et attributs
  - les méthodes spéciales : constructeur, destructeur
  - les surcharges d'opérateurs
  - l'héritage de classes
  - etc.

Python 3 – Programmation orientée objet

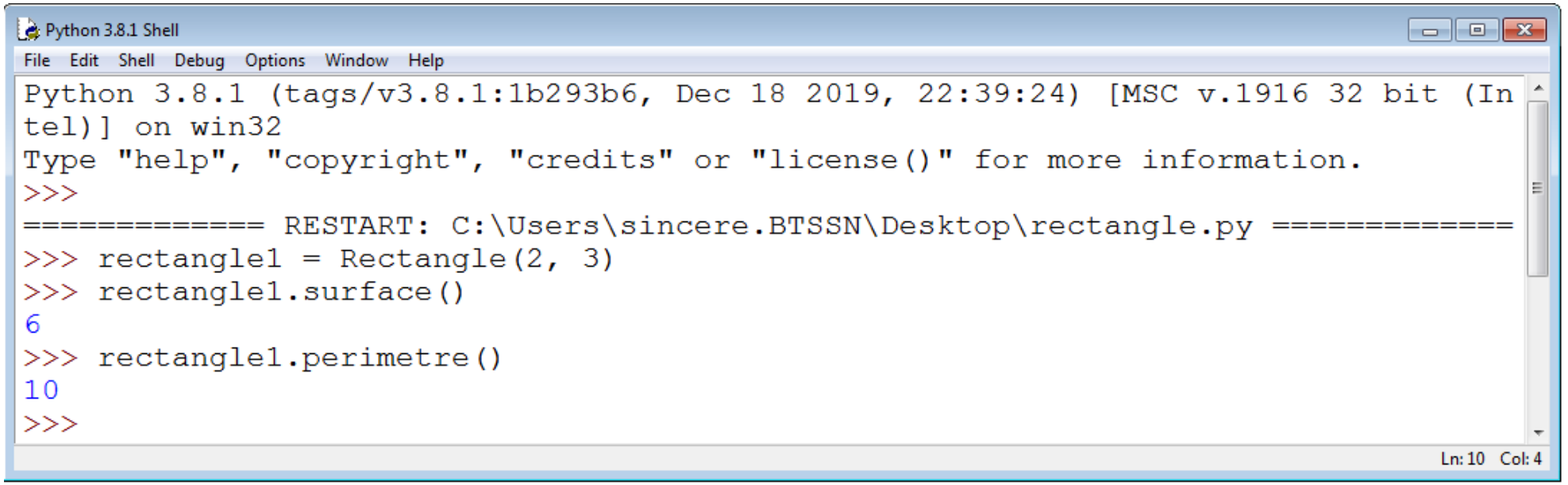
rectangle.py - C:\Users\sincere.BTSSN\Desktop\rectangle.py (3.8.1)

File Edit Format Run Options Window Help

```
class Rectangle:
    """ classe rectangle """

    # constructeur
    def __init__(self, longueur, largeur):
        # attributs
        self.long = longueur
        self.larg = largeur

    # méthodes
    def surface(self):
        return self.long*self.larg

    def perimetre(self):
        return 2*(self.long+self.larg)
```

Ln: 18 Col: 0

Python 3 – Programmation orientée objet



Python 2 versus Python 3

- print 'bonjour'
- a = 5/2

  donne un entier (2)
- raw_input()
- import Tkinter

- print('bonjour')
- a = 5/2

  donne un float (2.5)
- input()
- import tkinter

*Complément : objets non mutables

- Parmi les objets « immutables », nous avons les nombres (int, float), les chaînes de caractères (str) les tuples et les objets de type range.
- Les objets immutables ne peuvent pas être modifiés.

*Complément : objets mutables

- Parmi les objets mutables, nous avons les listes (list), les dictionnaires (dict), les instances de classe.

  Les objets mutables peuvent changer de valeur.

  Par contre, la référence reste inchangée (c'est l'adresse en mémoire de l'objet).

- Remarque : la clé d'un dictionnaire ne peut pas être mutable.

*Complément : la copie de listes pour les noobs

- Testons le code suivant :

```
>>> pierre = ['pierre.dupont@mail.com', '0612345678']
>>> sophie = pierre
>>> sophie[0] = 'sophie.martin@mail.com'
>>> sophie[1] = '0745678901'
>>> sophie
['sophie.martin@mail.com', '0745678901']
```

- Et maintenant, la surprise du chef (pour les noobs) :

```
>>> pierre
['sophie.martin@mail.com', '0745678901']
```

*Complément : la copie de listes pour les noobs

- Que se passe-t-il ?

  C’est le comportement attendu de Python :

  >>> sophie = pierre

  La variable « sophie » copie simplement la référence de la variable « pierre ».

  « sophie » et « pierre » ont donc la même référence, qui représente un seul et même objet (une liste, objet mutable).

  >>> sophie is pierre # opérateur de comparaison de 2 objets

  True

  Modifier « sophie » revient à modifier « pierre » et vice versa.

*Complément : la copie de listes pour les noobs

- Alors, quelle est la bonne technique pour copier une liste ?

```
>>> sophie = pierre[:]
>>> sophie = pierre.copy()    # autre écriture
>>> sophie is pierre
False
```

- N.B. Ce mécanisme de copie peut être insuffisant notamment si la liste contient des objets mutables.

  Il faut alors passer par une copie « profonde » (fonction deepcopy() du module copy).

- Même constat pour la copie d'un dictionnaire (lui aussi un objet mutable).

*Complément : encore des copies

- Intéressons-nous à présent à la copie d'objets non mutables (int, float, tuple) :

```
>>> a = 5
>>> b = a
>>> b is a
True
>>> b = b + 1
>>> b
6
>>> a   # la modification de b n'a pas d'influence sur a
5
>>> b is a
False
```

*Complément : encore des copies

- La variable « b » représente un entier, objet non mutable.

  Alors pourquoi puis-je modifier sa valeur ?

  En réalité, le contenu initial de « b » n'a pas été modifié car l'objet entier 5 est immutable : l'entier 5 est encore présent en mémoire à la même adresse.

  >>> b = b + 1

  La variable « b » conserve le même nom, mais c'est sa référence qui change : maintenant on pointe vers un objet entier 6, qui dispose d'un nouvel emplacement en mémoire.

- Même constat avec un float ou un tuple.

*Complément : erreurs à éviter

- Attention à ne pas choisir des noms de variables inappropriés :

```
def double(x):
        return 2*x
>>> y = double(10)
>>> y
20
>>> double = double(3)    # pourquoi pas ?
>>> double    # tout va bien
6
>>> a = double(5)     # ah ben non !
TypeError: 'int' object is not callable
```

*Complément : erreurs à éviter

- Où est le problème ?

```
def double(x):
        return 2*x
```

- Sans surprise « double » représente une fonction :

```
>>> double
<function __main__.double(x)>
>>> double = double(3)
>>> double   # double est maintenant un entier, on a écrasé la référence à la fonction
6
>>> a = double(5)    # cela n'a pas de sens : 6 n'est pas une fonction !
TypeError: 'int' object is not callable
```

*Complément : erreurs à éviter (2)

- Un autre exemple avec un nom de variable inapproprié :

```
>>> from math import *
>>> sin(pi/2)    # sinus de pi/2 (vaut exactement 1)
1.0
>>> pi = 4    # pourquoi pas ?
>>> sin(pi/2)
0.909297426825681 7
```

- Où est le problème ?

  Manque de chance, pi est une donnée du module math

```
>>> from math import *
>>> pi
3.141592653589793
>>> pi = 4          # on écrase la valeur de pi
>>> sin(pi/2)       # on fait maintenant n'importe quoi
0.909297426825681 7
```

*Complément : erreurs à éviter (2)

- On évite ce problème en important le module math de manière classique :

```
>>> import math
>>> math.sin(math.pi/2)     # oui, l'écriture est plus lourde
1.0
>>> pi = 4    # pourquoi pas ?
>>> math.sin(pi/2)
0.9092974268256817
>>> math.sin(math.pi/2)
1.0
```

- En définitive, pi et math.pi sont deux variables bien distinctes.

  La lourdeur de l'écriture (préfixe math.) est favorablement compensée par un cloisonnement des espaces de noms.

*Complément : erreurs d'arrondi

- Il faut bien garder à l'esprit que le calcul sur les nombres flottants (type float) se fait avec 16 chiffres significatifs :

```
>>> 1e15 + 1 -1e15
1.0
>>> 1e16 + 1 -1e16    # résultat attendu : 1
0.0
>>> (2**0.5)**2     # résultat attendu : 2
2.0000000000000004
>>> 0.1 + 0.2 == 0.3     # résultat attendu : True
False
```

*Complément : erreurs d'arrondi

- Ainsi, pour comparer deux nombres flottants, il faut tenir compte des éventuelles erreurs d'arrondi, et s'interdire l'opérateur d'égalité ==

```
def egale(x, y):
        # return x == y     # à proscrire pour les floats
        return abs(x-y) < 1e-9   # on se fixe une tolérance
>>> egale(0.1+0.2, 0.3)
True
```

- A voir : fonctions fsum() et isclose() du module math

*Complément : une fonction qui retourne une … fonction

- Une fonction peut retourner n'importe quel type d'objet, en particulier une fonction :

```
def affine(a, b):
        def f(x) :     # fonction locale
            return a*x + b
        return f
>>> droite1 = affine(2, 1)
>>> droite1
<function __main__.affine.<locals>.f(x)>
>>> droite1(7)   # retourne 2*x + 1
15
```

*Complément : une fonction qui prend comme argument une fonction et qui retourne une fonction

- Une fonction peut prendre en argument n'importe quel type d'objet, en particulier une fonction :

```
def carre(f):
        def g(x) :     # fonction locale
            return f(x)**2
        return g
>>> import math
>>> cos2 = carre(math.cos)
>>> cos2
 <function __main__.carre.<locals>.g(x)>
>>> cos2(0.1)   # retourne cos(x) au carré
0.990033289206209
```

Une petite conclusion

Il y a encore beaucoup de concepts à maîtriser avant de devenir un expert en Python (je n'ai pas cette prétention).

En vrac : les itérateurs, générateurs, décorateurs, les espaces de noms, les contextes managers… et autres diableries.

En dehors de la compréhension de la logique interne de Python, il faudra vous investir dans l'utilisation des modules liés à vos centres d'intérêts (par exemple tkinter pour faire une interface graphique, numpy pour faire du calcul numérique…).

Pour les électroniciens et les adeptes des objets connectés, vous trouverez votre bonheur avec les cartes Raspberry Pi.

Sachez aussi qu'il existe MicroPython, une implémentation de Python pour microcontrôleur qui tourne avec quelques dizaines de ko de mémoire vive !

Copyright

Document réalisé par Fabrice Sincère.

Pour toutes remarques ou suggestions :

Fabrice.sincere@wanadoo.fr

- Téléchargement :

  http://fsincere.free.fr/isn/python/download/diaporama/diaporama_presentation_python3.pdf
- Bibliographie :

  http://fsincere.free.fr/isn/python/cours_python.php
- Version du document : 2.3 (mars 2020)
-