Brève présentation du document

- La formation a concerné une vingtaine de collègues, issus principalement de deux lycées.
- Pour des raisons matérielles liées à l'infrastructure réseau du LGT Maupassant, c'est une distribution de python 3 avec spyder qui a été déployée.
- Une deuxième séance est programmée pour février 2018.
- Le début du document peut être modifié pour servir de séance de première découverte de python avec des élèves (c'est d'ailleurs le chemin inverse que ce document a connu durant son élaboration).
- Ce document se veut complémentaire des documents fournis sur Eduscol et par le pôle de compétences TICE de l'académie, et notamment à destination de collègues découvrant complètement l'utilisation d'interfaces et de langages du type de python.

Vous avez découvert l'algorithmique au collège à travers le langage scratch.
Vous continuerez votre étude de l'algorithmique tout au long du lycée avec le langage **python** *et son interpréteur* **spyder**.

Table des matières

I Nature du langage python

Le langage python est un **langage interprété** : vous saisirez du code ligne par ligne, qui sera transformé ligne par ligne en langage machine, puis exécuté par la machine si nécessaire.

Allumez le PC. Recherchez le logiciel spyder et lancez-le. Ce logiciel propose divers environnements de programmation : celui qui s'ouvre est IDLE. Observez bien l'organisation des fenêtres.

- Fenêtre console : c'est la console qui vous permettra de saisir du code qui sera ensuite exécuté par l'interpréteur python. On valide chaque commande par la touche entrée, et si nécessaire, un message d'erreur ou un résultat s'affiche dans la console.
- Fenêtre éditeur de fichiers : la console présente l'inconvénient de ne pas conserver la trace du code saisi une fois le logiciel spyder quitté. Pour ne pas avoir à tout resaisir, on peut stocker du code dans des fichiers `.py`.

  Pour exécuter le code contenu dans un fichier `.py` :
  – Ouvrir le fichier `.py` : menu Fichier>Ouvrir
  – Exécuter : menu Exécution>Exécuter (F5)

II Utiliser python comme une calculatrice

1 Les quatre opérations

Opération	Syntaxe	Exemples
Addition	`+`	`7+3`
Soustraction	`-`	`7-3` `3-7`
Multiplication	`*`	`7*3`
Division	`/`	`7/3`
Quotient	`//`	`7//3`
Reste	`%`	`7%3`

$$\begin{array}{rl|l} & 7 & 3 \\ - & 2\times 3 & 2 \\ \hline & 1 & \end{array}$$

$7/3 \simeq 2,33333$

Attention : en python 2, le code `7/3` calcule `7//3`. Le passage en `float` doit être forcé en saisissant `7./3`.

2 Puissances et priorités

Opération	Syntaxe	Exemples
Puissance	`**`	`2**3` pour 2^3, soit 8
Notation scientifique	$\pm$*mantisse* `e` *exposant*	`-3e5` pour -3×10^5, soit $-30\,000$ `3e-5` pour 3×10^{-5}, soit $0,00003$

a) Le code `2**3**4` calcule-t-il $2^{(3^4)}$ ou $(2^3)^4$?
Code pour $2^{(3^4)}$: ..... Code pour $(2^3)^4$: .....
Code pour 2^{34} : .....

b) Le code `-3**2` calcule-t-il 9 ou -9 ?
Code pour $(-3)^2$: ..... Code pour $-(3^2)$: .....
Les ..... sont prioritaires par rapport aux .....

c) Calculer $2(3+5)$ à la main. On obtient .....
Saisir le code `2(3+5)`. Que se passe-t-il, et pourquoi ?
.....
.....
Code correct pour calculer $2(3+5)$: .....

d) Associer les codes aux calculs

$2+\frac{3}{5}+7$ •	• `(2+3)/(5+7)`
$\frac{2+3}{5}+7$ •	• `2+3/(5+7)`
$2+\frac{3}{5+7}$ •	• `2+3/5+7`
$\frac{2+3}{5+7}$ •	• `(2+3)/5+7`

e) Que calcule `2/5*8` ?
Code pour $\frac{2}{5\times 8}$: ..... Code pour $\frac{2}{5}\times 8$: .....

3 Et si on calculait avec du texte ?

Opération	Syntaxe	Exemples
Texte	Encadrer avec ' ou avec "	`'bonjour'` `"bonjour"`

En informatique, le texte est représenté par ce qu'on appelle *des chaînes de caractères.*
Les caractères constitutifs d'une chaîne sont numérotés *en même temps* de gauche à droite (0, 1, 2, ...) et de droite à gauche (-1, -2, -3, ...).

Tester les codes suivants :

- `'bonjour'[0]` .....
- `'bonjour'[1]` .....
- `'bonjour'[2]` .....
- `'bonjour'[4]` .....
- `"bonjour"[2:4]` .....
- `"bonjour"[:4]` .....
- `'bonjour'[-1]` .....
- `'bonjour'[-2]` .....
- `'bonjour'[-3]` .....
- `'bonjour'[-6]` .....
- `"bonjour"[2:]` .....
- `"bonjour"[:]` .....

Qu'arrive-t-il avec le code `'bonjour'[7]` ?

.....

.....

Expliquer l'effet du code `len('bonjour')`

.....

D'où vient le nom de la commande `len` ?

.....

Python attribue un sens différent aux opérations,
selon le type de donnée auquel elles s'appliquent.

Compléter le tableau ci-dessous.

Code	Résultat	Explication
`type('27')`		
`type(27)`		
`type(27.)`		
`'27'+'3'`		
`27+3`		
`27.+3.`		
`'27'+3`		
`int('27')+3`		

4 Fonctions usuelles

Tester les codes ci-dessous

```
sqrt(2)  pi  sin(pi)  cos(pi/3)
```

Que se passe-t-il ?

.....

4 Fonctions usuelles (suite)

Python ne connaît pas de valeur approchée du nombre π, ni les fonctions racine carrée $\sqrt{x}$, ni les fonctions sinus et cosinus...
Nous allons devoir lui apprendre tout cela, mais comme nous n'allons pas tout reprogrammer de A à Z, nous allons charger du code python préexistant au sein d'un fichier appelé *module*.
On peut importer un module de plusieurs façons. Aujourd'hui, voyons la plus simple :

Exécuter le code `from math import *`, puis réesayer les codes

`sqrt(2)` `pi` `sin(pi)` `cos(pi/3)`

Que se passe-t-il ?
.....
Comment s'appelle le module qui contient les fonctions usuelles ?
.....

Fonction	En python
$\sqrt{x}$	`sqrt(x)`
$\|x\|$	`abs(x)`
$\sin(x), \cos(x), \tan(x)$	`sin(x),cos(x),tan(x)`
$\sin^{-1}(x), \cos^{-1}(x), \tan^{-1}(x)$	`asin(x),acos(x),atan(x)`
$\ln(x)$	`log(x)`
$\log(x)$	`log(x,10)`
e^x	`exp(x)`
e, π	`e,pi` ATTENTION : `e=13` est licite. Recharger le module math pour retrouver la valeur initiale attribuée à `e`.

Conversions
`radians(180)`
`degrees(pi)`

III Interface et syntaxe

1 Généralités

- Pour exécuter plusieurs instructions au sein d'une même ligne de code, séparer les instructions par un point virgule.

```
5-2 ; 3+5
```

  Si seul le résultat de `3+5` apparaît, on pourra proposer d'éxécuter `(5-2,3+5)`.
- Affectation d'une variable :

```
a=2
```

- Affichage du contenu de la variable `a` dans la console :

```
a
```

  On peut aussi utiliser l'explorateur de variables.
- Affectation de plusieurs variables d'un coup

```
a,b=2,3
```
Comparer `a,b=(2,3)` et `a=(2,3)`.

- Le typage des variables est dynamique : il peut changer à chaque nouvelle affectation.

```
a=2
type(a)
a="bonjour"
type(a)
```

- Transtypage : `int("123")` renvoie l'entier 123.
- Python est sensible à la casse : `a` et `A` ne renvoient pas aux mêmes variables.

```
a=1 ; A=2
a
A
```

- Identifiant de l'adresse en mémoire d'une variable `a`

```
id(a)
```

- On commente une ligne avec #, et un ensemble de lignes en l'encadrant de triples guillemets """*bla bla bla*""".
- Blocs : ils sont matérialisés par l'indentation (tabulations). Pas de begin/end à gérer, mais attention aux surprises !
  L'annonce de l'ouverture d'un bloc se fait avec le double point " :".

Comparer les trois codes

```
def Test1(x):
    if x==2:
        return "C'est 2"
    return "Ce n'est pas 2"

def Test2(x):
    if x==2:
        2+2
    return "C'est 2"
    return "Ce n'est pas 2"

def Test3(x):
    if x==2:
    return "C'est 2"
    return "Ce n'est pas 2"
```

- Définition d'une fonction

```
def NomFonction(param1,param2,...,paramn):
    code code code
    return Resultat
```

– Variable de type fonction
(pratique pour passer une fonction numérique à étudier en paramètre)

```
f=lambda x:3*x+2
f(1)
from math import *#pour sqrt()
f=lambda x,y:sqrt(x**2+y**2)
f(1,1) ; sqrt(2)
```

– Disjonction de cas

```
if Prédicat:
    code code code
else:
    code code code
```

On imbrique des tests avec `elif` *Prédicat*: (pas de commande du type `switch/case`).

– Boucles. On pourra se référer au document ressource *Algorithmique et programmation* disponible sur Eduscol, page 4. Quelques instructions à tester :

```
range(5)
range(2,5)
range(1,11,2)
[i for i in range(10)]
[i for i in range(10) if i%2==1]
[i for i in range(10) if int(sqrt(i))==sqrt(i)]
```

2 Principaux types

– Nombres :
  – Entiers `int`
  – Flottants `float`
  – Complexes `complex`
– Listes
  – Listes `list` Exemple : `[2,3,4,'uu']`. Les composantes sont numérotées de 0 à `len(`*liste*`)-1`. On peut imbriquer des listes `[[1,2],[3,5,6],[[4],[5]]]`. Les composantes des listes sont modifiables : `L[0]=1` avec `L=[2,3]`.
  – tuples `tuples` Exemple : `(2,3,4)`. Les composantes des tuples ne sont pas modifiables.
– chaînes de caractères `str`
– Booléens `bool` valant `True` ou `False`

3 Logique

Les opérateurs logiques sont `and`, `or` et `not`.
Les tests sont

Test	En python
Egalité	`==`
Non égalité	`!=`
$\leqslant$	`<=`

et leurs variations naturelles.

4 Modules

Il existe de très nombreux modules.
En voici quelques'uns qui permettent de couvrir à peu près l'ensemble des programmes du lycée.
Différentes façons de charger tout ou partie d'un module et code pour accéder à ses fonctions :

`import` *module*	*module*`.`*fonction*`(`*arguments*`)`
`from` *module* `import` *fonction*	*fonction*`(`*arguments*`)`
`from` *module* `import *`	*fonction*`(`*arguments*`)`
`import` *module* `as` *alias*	*alias*`.`*fonction*`(`*arguments*`)`

- La commande `Fraction` du module `fractions` permet de travailler avec des nombres rationnels.
- `math` contient la plupart des fonctions et commandes mathématiques classiques.
- `numpy` (alias habituel `np`) propose des commandes pour le calcul numérique.
- `scipy` (alias habituel `sp`) permet d'effectuer des calculs formels.
  Se rendre sur `docs.scipy.org` pour une documentation complète de `numpy` et `scipy`.
- `matplotlib` (alias habituel `mpl`) permet le tracé de graphiques. On utilisera le plus souvent son sous-module `pyplot` chargé par `from matplotlib import pyplot as plt`.
- `random` permet de simuler des nombres pseudo aléatoires.
- `copy` permet de réaliser des copies d'objets en profondeur, comme `CopierObjetLibre(<`*Objet*`>)` sous géogébra. Faites des expériences avec la commande `id(`*variable*`)` qui donne l'identifiant de l'adresse en mémoire d'une variable et la commande `deepcopy(`*variable*`)` qui réalise une copie en profondeur.

`dir(`*module*`)` liste les fonctions du module *module*.

IV Précision des calculs et systèmes de numération

On retrouve en python toutes les particularités du calcul numérique.

1 Écrire en base 10 des nombres stockés en base 2 en un nombre fini de bits

Certains nombres décimaux s'écrivent bien en base 2, d'autres non.

Base 10	Base 2	Troncature sur 10 bits	Explication
$0,1$	$0,0\overline{0011}$	$0,0001100110$	$\frac{1}{10}=\frac{1}{2}\times\frac{1}{5}$ (voir $0,2$)
$0,2$	$0,\overline{0011}$	$0,0011001100$	$\frac{1}{5}=\frac{3}{16}+\frac{1}{16}\times\frac{1}{5}$
$0,3$	$0,01\overline{0011}$	$0,0100110011$	$\frac{3}{10}=\frac{1}{4}+\frac{1}{4}\times\frac{1}{5}$
$0,1+0,2$		$0,0100110010$	

d'où le gag `0.1+0.2`, qui peut être évité en utilisant le module `decimal` ou la commande `fractions.Fraction`, et qu'on peut expliquer à l'aide du tableau précédent.

2 Limites de l'écriture à 15 décimales et tests d'égalités

Le gag suivant fonctionne aussi sur Casio 35 et sur TI 83.

```
1-1+1e-16 ; 1+1e-16-1
```

Cela pose le problème de la validité des tests d'égalités.

```
1==1+1e-16
```

On peut contourner le problème à l'aide d'une fonction

```
def SontEgaux(x,y):
   return abs(x-y)<1e-16
```

ou si $|x|$ n'est pas encore connu des élèves

```
def SontEgaux2(x,y):
   return (-1e-16<x-y) and (x-y<1e-16)
```

mais c'est au prix de la perte de la transitivité de l'égalité.

```
SontEgaux(1e-17,6e-17)
SontEgaux(6e-17,1.1e-16)
SontEgaux(1e-17,1.1e-16)
```

Ce phénomène se produira aussi plus ou moins facilement selon le DL éventuel des fonctions utilisées :

```
cos(1e-8)
mais
sin(1e-127)
```

V Représentations graphiques

1 Obtention d'un graphique

Document du pôle de compétence pages 7 et 36.

2 Tracé de la courbe représentative d'une fonction

Document du pôle de compétence page 13.

VI Quelques thèmes des programmes

1 Probabilités

On utilise le module `random`

Syntaxe	Explication
`randint(`a`,`b`)`	Tirage pseudo équiprobable d'un entier entre a et b.
`random()`	Tirage pseudo uniforme d'un flottant entre 0 et 1.
`uniform(`a`,`b`)`	Tirage pseudo uniforme d'un flottant entre a et b.

La fonction `uniform(`a`,`b`)` équivaut à `a+(b-a)*random()`.

1.1 Simuler des lancers indépendants

```
"""Simulations classiques en probabilites"""

from random import *

##Simulation d'experiences aleatoires
def JetPieceEquilibree():
    if randint(0,1)==0:#Tirage pseudo-equiprobable sur {0,1}
        return "Pile"
    else:
        return "Face"

def JetPieceDesequilibree(p):
    if (0<=p) and (p<=1):
        if random()<p:
            return "Pile"
        else:
            return "Face"
    else:
        return "p doit etre situe entre 0 et 1"

def JetDeEquilibre():
    return randint(1,6)

def JetDePipe(probas):
    #probas est cense etre une liste de 6 reels positifs de somme 1
    #Calcul des seuils de changement de face du de
    Seuils=probas
    for i in range(1,6):
        Seuils[i]=Seuils[i]+Seuils[i-1]
    #Seuils[5]=1 #Dans un deuxieme temps, apres observation

    #Tirage pseudo-uniforme d'un reel entre 0 et 1
    Tirage=random()

    #Identification de la face du de tiree
    for i in range(6):
        if Tirage<=Seuils[i]:
            return (i+1,Seuils,Tirage)#Dans un premier temps, pour comprendre
            #return i+1#Dans un deuxieme temps, apres observation

#Compter le nombre de 'Pile'
#apres n lancers independants d'une piece desequilibree
def NombreDePileApresNLancers(n,p):
    Tirages=[]
    for i in range(n):
        Tirages.append(JetPieceDesequilibree(p))
    return Tirages.count("Pile"),Tirages
```

```
#Observer la stabilisation de la frequence de 'Pile'
#lors de repetitions independantes du lancer d'une meme piece desequilibree
def  StabilisationFrequenceNLancersMemePieceDesequilibree(n,p):
    NombreDeTiragesDejaRealises=0
    NombreDePileObserves=0
    FrequencesDePileAuCoursDesLancers=[]
    for i in range(n):
        NombreDeTiragesDejaRealises=NombreDeTiragesDejaRealises+1
        if JetPieceDesequilibree(p)=='Pile':
            NombreDePileObserves=NombreDePileObserves+1
        FrequencesDePileAuCoursDesLancers.append(
            float(NombreDePileObserves)/NombreDeTiragesDejaRealises)#Transtypage
            #inutile en python 3 mais imperatif en python 2,
            #sinon c'est la division euclidienne qui sera effectuee.
    return FrequencesDePileAuCoursDesLancers[-1]#Pour ne renvoyer que la derniere
    #frequence calculee (bien plus rapide)
    #return FrequencesDePileAuCoursDesLancers#Pour tout renvoyer

#Tracer la courbe de stabilisation des frequences
#--> voir partie sur la representation de courbes de fonctions
```

On pourra aussi se référer au document ressource *Algorithmique et programmation* disponible sur Eduscol, pages 13 à 23.

1.2 Simuler une urne

```
"""Simulation d'une urne contenant deux boules bleues et une boule rouge"""

from random import *
from copy import *

Couleurs={'b':'bleue','r':'rouge','n':'noire','v':'verte'}
#Les dictionnaires {clef:valeur, clef:valeur,...}
#c'est de la decoration, hors programme

def InitialiserUrne():
    return ['b','b','r']

def Couleur(boule,urne=InitialiserUrne()):
    #si le parametre urne n'est pas precise,
    #on utilise l'urne par defaut generee par InitialiserUrne()
    return Couleurs[urne[boule]]

def TirageAvecRemise(urne=InitialiserUrne()):
    if len(urne)==0:
        return "Urne vide"
    else:
        Tirage=randint(0,len(urne)-1)
        #return (Couleur(Tirage,urne),Tirage)#Dans un premier temps, pour comprendre
        return Couleur(Tirage,urne)#Dans un deuxieme temps
```

```
def TiragesAvecRemise(NombreDeTirages,urne=InitialiserUrne()):
    Tirages=[]#Liste vide avant le premier tirage
    for i in range(NombreDeTirages):
        Tirage=TirageAvecRemise(urne)#On simule un tirage avec remise
        Tirages.append(Tirage)#On ajoute la couleur de la boule tiree
    return Tirages

def TirageSansRemise(urne=InitialiserUrne()):
    if len(urne)==0:
        return ("Urne vide",[],"Aucune boule tiree")#[] represente une urne vide
    else:
        NumeroBouleTiree=randint(0,len(urne)-1)
        LaCouleur=Couleur(NumeroBouleTiree,urne)
        urne=deepcopy(urne)#Ici, demander une explication
        #au tableau pour tous a Nicolas
        urne.__delitem__(NumeroBouleTiree)
        return (LaCouleur,urne,NumeroBouleTiree)
    #Renvoie un tuple constitue de la couleur tiree, de l'urne apres tirage
    #et du numero de la boule tiree, pour comprendre.

def TiragesSansRemise(NombreDeTirages,urne=InitialiserUrne()):
    Tirages=[]#Liste vide avant le premier tirage
    for i in range(NombreDeTirages):
        Tirage,urne,NumeroTire=TirageSansRemise(urne)#On simule un tirage
        #sans remise et on modifie l'urne suite au tirage
        Tirages.append(Tirage)#On ajoute la couleur de la boule tiree aux resultats
    return Tirages


#Compter le nombre de boules d'une couleur donnee
#apres n tirages avec remise dans l'urne par defaut
def NombreDeBoulesBleuesApresNTiragesAvecRemise(n):
    Tirages=TiragesAvecRemise(n)
    return Tirages.count("bleue")
```

1.3 Simuler des lois

Sous-module `random` du module `numpy`.

1.4 Aires, probabilités et fréquences

```
#Approximation de pi/4

from math import *
from random import *

print("pi/4="+str(pi/4))

def PointDansCarre():
    return (random(),random())
```

```
def EstDansLeDisqueUnite(point):
    return sqrt(point[0]**2+point[1]**2)<=1

def ApprocherPiSurQuatre(n):
    score=0
    for i in range(n):
        if EstDansLeDisqueUnite(PointDansCarre()):
            score=score+1
    return float(score)/n#transtypage inutile en python 3
    #mais imperatif en python 2
```

1.5 Statistiques à une ou plusieurs variables

Sous-module `statistics` du module `numpy`. Ce sera l'occasion de travailler un peu l'anglais : `average` pour moyenne etc. On pourra aussi se référer au document ressource *Algorithmique et programmation* disponible sur Eduscol, pages 7 et 8.

1.6 Générer ou charger des grandes séries statistiques

(Merci à Stéphane Gentil pour ces scripts)

```
#Lecture/ecriture d'un fichier de donnees statistiques au format csv

import os

f=open("D:/DataStat.csv","r")
L=[]
for l in f:
    ligne_tableau=l.rstrip("\n").split(",")#"\t" tabulation
    #Ce serait "," pour virgule, "\n" pour retour charriot etc
    L.append(ligne_tableau)
f.close()

#reste a transtyper en float les valeurs,
#qui sont pour l'instant des str

for i in range(len(L)):
    for j in range(len(L[i])):
        L[i][j]=float(L[i][j])

#Et le chemin inverse...
for i in range(len(L)):
    for j in range(len(L[i])):
        L[i][j]=str(L[i][j])

f=open("D:/DataStat2.csv","w")
for ligne in L:
    ligne_texte="\t".join(ligne)
    f.write(ligne_texte+"\n")
f.close()
```

2 Une idée de sujet pour une séance en salle informatique en seconde en géométrie repérée

```
""" (texte commente encadre de triple guillements)
Sujet de seance informatique.
1) Completez les definitions des fonctions
2) Proposez une sequence de lignes de code qui prouve que le quadrilatere
ABCD de sommets A(1,2) B(4,4) C(6,8) et D(3,6) est un parallelogramme non losange.

Les coordonnees sont toutes lues dans un meme repËre orthonormÈ (O;I,J).
"""
from math import * #necessaire pour appeler sqrt()

def Distance(x1,y1,x2,y2):
    #Renvoie la distance entre les points de coordonnees (x1,y1) et (x2,y2)
    return 0 #Un commentaire est precede de #

def EstVerticale(x1,y1,x2,y2):
    #Renvoie True quand la droite definie par les points (x1,y1) et (x2,y2)
    #est verticale, False sinon.
    return 0

def CoefficientDirecteur(x1,y1,x2,y2):
    #Renvoie le coefficient directeur de la droite
    #definie par les points (x1,y1) et (x2,y2), quand celle-ci n'est pas verticale.
    #Renvoie la chaine "verticale" quand la droite est verticale
    return 0

def OrdonneeALorigine(x1,y1,x2,y2):
    #Renvoie l'ordonnee a l'origine
    #de la droite definie par les points (x1,y1) et (x2,y2),
    #quand celle-ci n'est pas verticale.
    return 0

def SontParalleles(a1,b1,a2,b2):
    #Renvoie True quand les droites non verticales
    #d'equations y=a1*x+b1 et y=a2*x+b2 sont paralleles, False sinon.
    #Doit aussi renvoyer True quand les deux droites sont verticales
    #(a1=a2="verticale"),
    #et False quand une droite est verticale et pas l'autre.
    return 0

## Ci-apres le code qui prouve que le quadrilatere
#ABCD de sommets A(1,2) B(4,4) C(6,8) et D(3,6) est un parallelogramme non losange.
```

3 Calcul formel

On pourra se référer aux pages 35 et 36 de la formation python du 13 juin 2017 de messieurs Bourneuf, Labbé et Winspeare (académie de Nantes), disponible en ligne.

4 Algèbre linéaire en spe TS

Sous-module `linalg` du module `numpy`.

5 Intégration

Sous-module `integrate` du module `scipy`.

6 Tortue

Le lien avec scratch au collège est bien détaillé dans le document du pôle de compétence et dans le document ressource *Algorithmique et programmation* disponible sur Eduscol page 6.

```
#Une marche aleatoire de tortue

import turtle
from random import *

def PasVersLaGauche():
    turtle.left(90)
    turtle.forward(10)

def PasVersLaDroite():
    turtle.right(90)
    turtle.forward(10)

def MarcheAleatoire(n):
    turtle.clear()
    turtle.down()
    for i in range(n):
        Tirage=randint(0,1)
        if Tirage==0:
            PasVersLaGauche()
        else:
            PasVersLaDroite()
    turtle.up()
```

7 Nombres complexes

Le nombre i est représenté par `1j`, le code `j` renvoyant une erreur.
Tout est objet en python. Vous constaterez en saisissant `dir(1+1j)` que le conjugué, la partie réelle et la partie imaginaire s'obtiennent ainsi :

```
(1+1j).conjugate()
(1+1j).real
(1+1j).imag
```

et que le module s'obtient indifféremment par `(1+1j).__abs__()` ou `abs(1+1j)`.
Pour obtenir un argument, il faut utiliser le module `cmath` et sa commande `phase`.

8 Suites

```
#Suites
import matplotlib.pyplot as plt

def u(n):
    return n**2+1

def PremiersTermes(suite,n):
    return [suite(i) for i in range(n)]

def TracerPremiersTermes(suite,n):
    X=[i for i in range(n)]
    Y=PremiersTermes(suite,n)
    plt.plot(X,Y,bo)

def v(n):
    if n==0:
        return 1
    vn=1
    for i in range(n):
        vn=vn**2+1
    return vn
```