Algorithmique et programmation

Plan

1. **L’introduction de Python en Seconde**
2. Les types de variables
3. Une progression en Seconde
4. Les listes
5. Premières séances en Première
6. La trace écrite
7. L’évaluation

1 L’introduction de Python en Seconde

Difficultés pédagogiques rencontrées lors de l’introduction de Python en Seconde ?

1 L'introduction de Python en Seconde

- La difficulté des types :

```
x = int(input("Valeur de x ? "))
double = 2*x
print("Le double de " + str(x) + " est " + str(double))
```

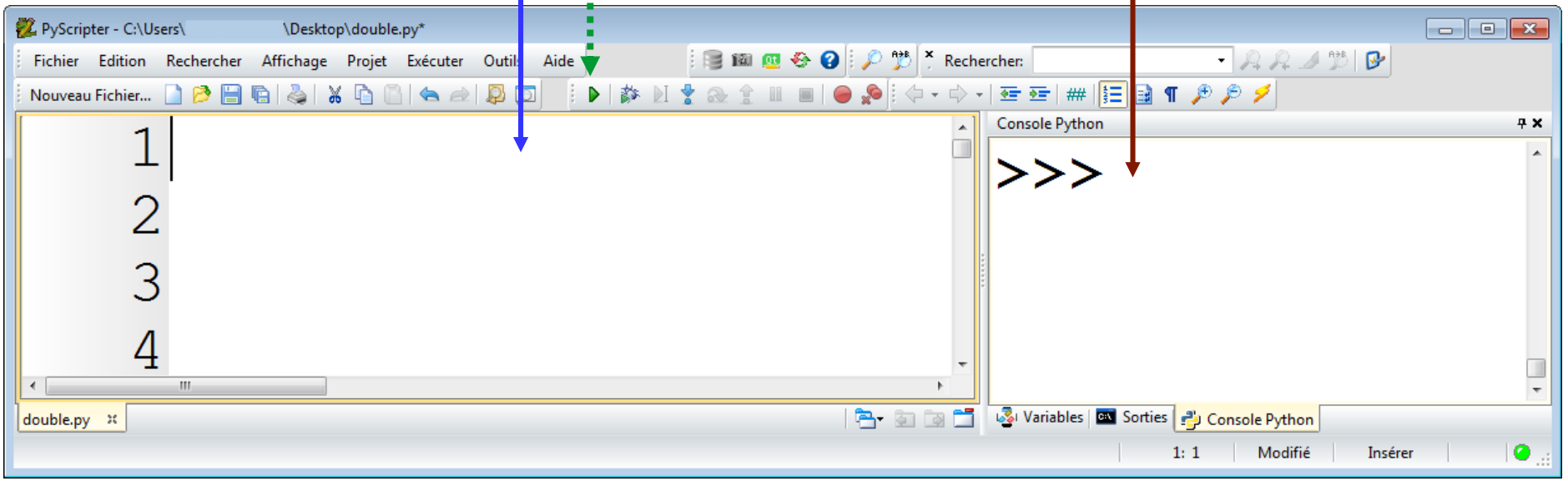
1 L'introduction de Python en Seconde

Environnement de travail Python :

Console

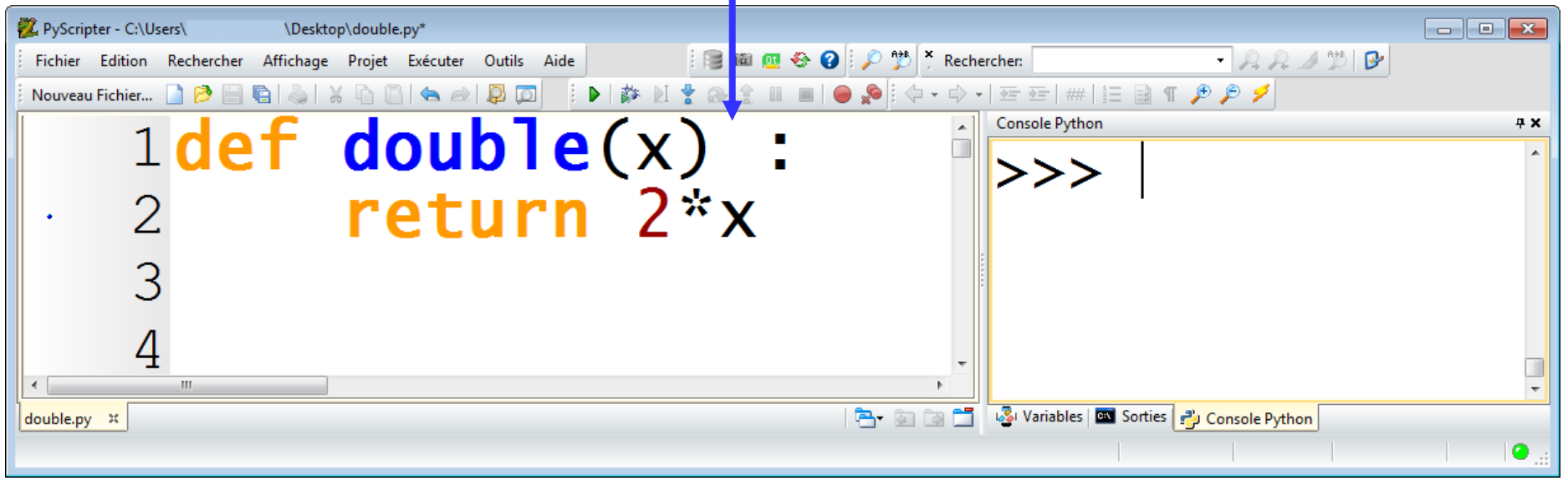
Éditeur

Bouton Exécuter



1 L'introduction de Python en Seconde

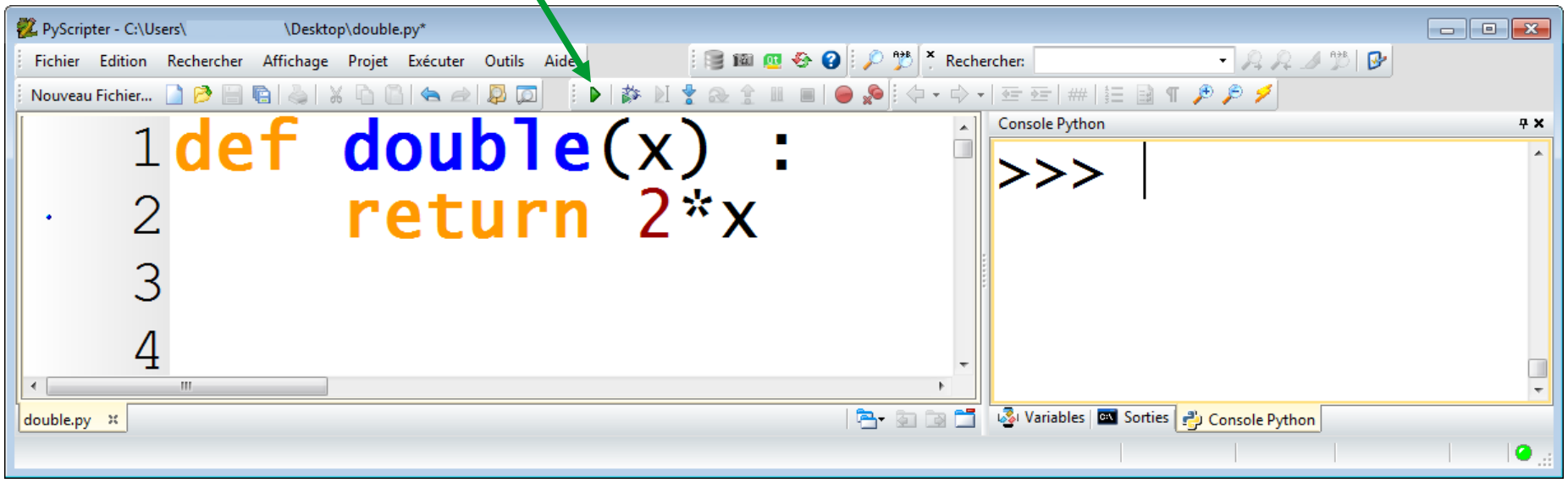
a) Écrire une fonction dans l'**éditeur** :



1 L'introduction de Python en Seconde

a) Écrire une fonction dans l'**éditeur** :

b) Lancer sa **Lecture :**

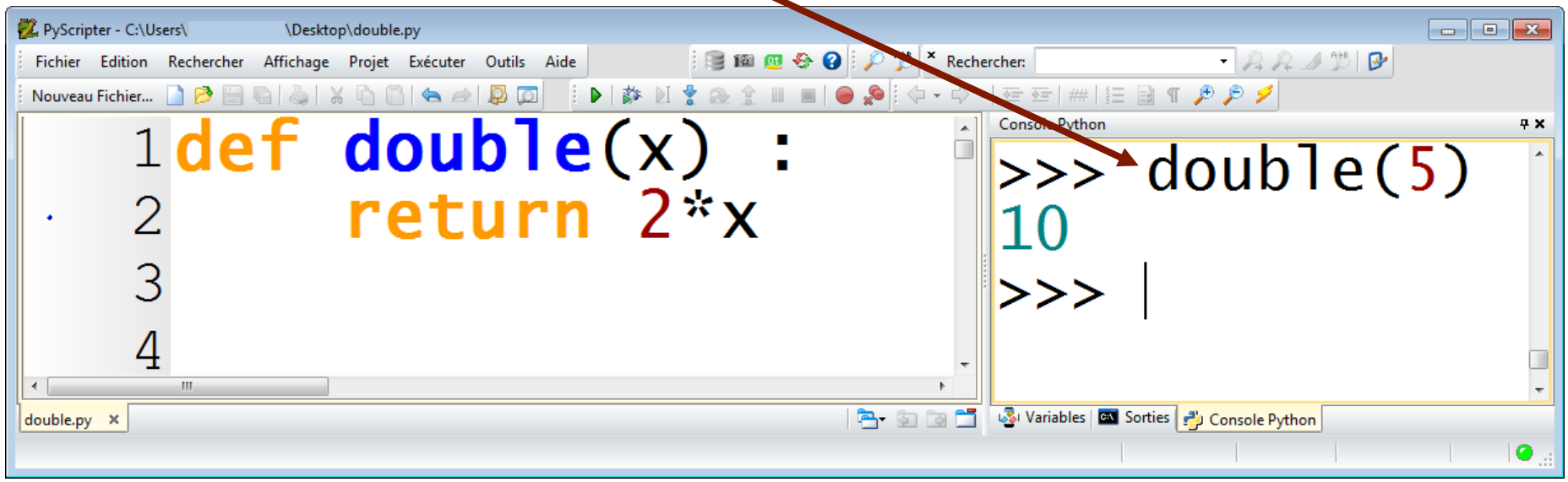


1 L'introduction de Python en Seconde

a) Écrire une fonction dans l'**éditeur** :

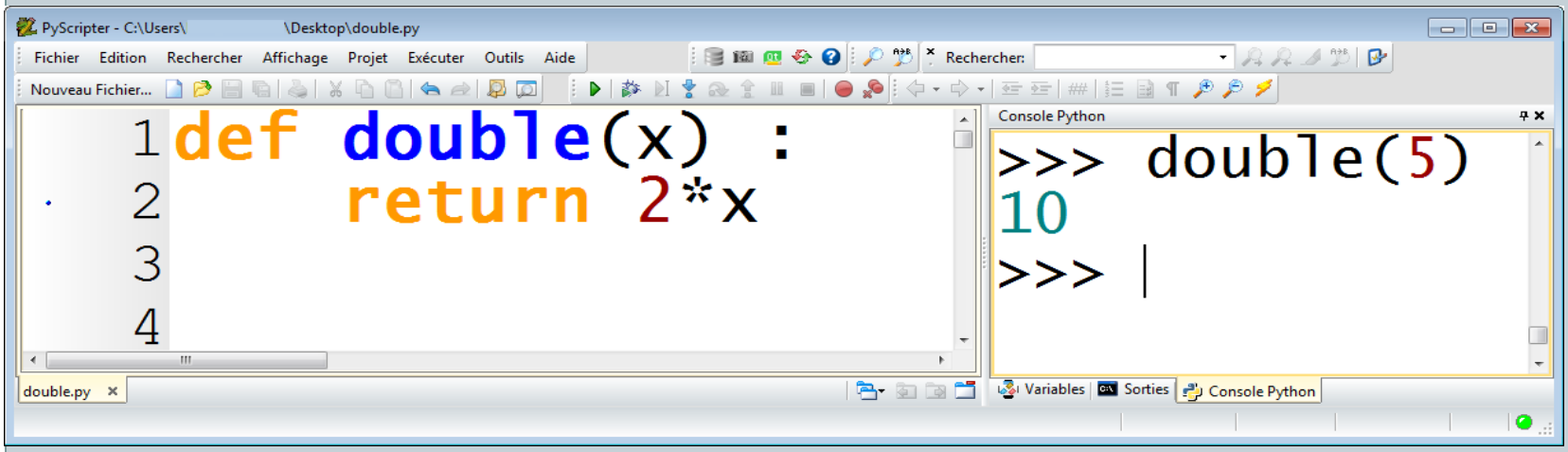
b) Lancer sa **Lecture** :

c) L'appeler depuis la **Console** :

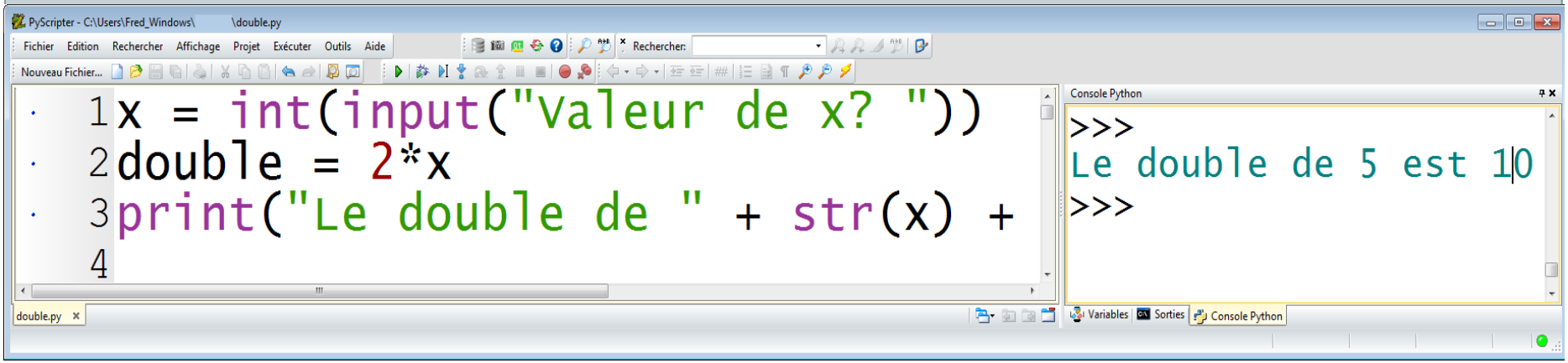


1 L'introduction de Python en Seconde

- Avec une **fonction** et la **console** :



- Avec input et print :



1 L'introduction de Python en Seconde

Avec une **fonction** et la **console** :

- on contourne la difficulté des types,

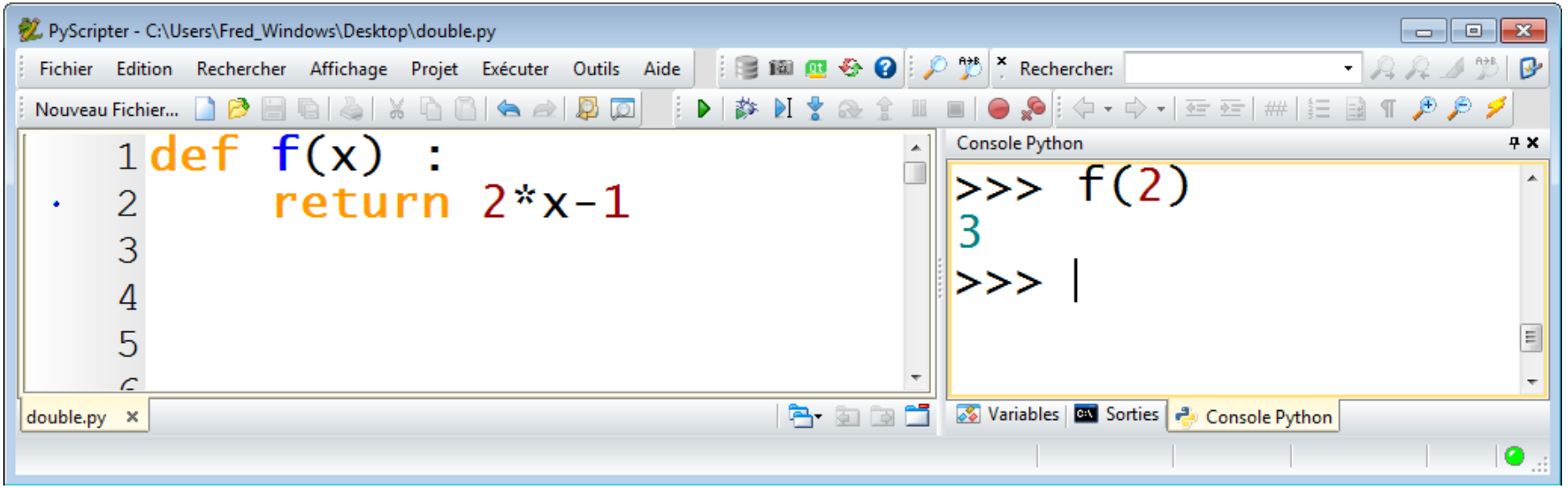
- on dispose d'une fonction qui peut être ré-exploitée dans la suite du programme, par une autre fonction par exemple : on divise ainsi un problème complexe en sous-problèmes plus simples.

1 L'introduction de Python en Seconde

De quels « points d'appuis pédagogiques », disposons-nous dans cette approche de Python ?

1 L'introduction de Python en Seconde

Un point d'appui pédagogique :



1 Une introduction de Python en Seconde

Ressource 1 : AlgoProg_Ressource_1_TP1.pdf

- **Objectifs :**
  - environnement python
  - la console
  - les variables informatiques
  - comparaison avec les variables mathématiques
  - algorithmes

- **Modalités :**
  - en salle informatique
  - 1 à 1,5 heure selon les élèves
  - sujet numérique (pas d'impression papier)
  - trace écrite : diaporama section 6

1 Une introduction de Python en Seconde

Ressource 2 : AlgoProg_Ressource_2_TP2.pdf

- **Entrée via :**
  - les fonctions, au sens mathématique
- **Objectifs :**
  - les fonctions informatiques
  - analogies et différences avec les fonctions mathématiques
  - utilisation d'une librairie de Python (math)
- **Modalités :**
  - en salle informatique
  - 1 à 1,5 heure selon les élèves
  - sujet numérique (pas d'impression papier)
  - trace écrite : diaporama section 6

1 L'introduction de Python en Seconde

- **Programme de Seconde :**

Algorithmique et programmation

[...]

Une consolidation des acquis du cycle 4 est proposée autour de deux idées essentielles :

- la notion de fonction ;
- la programmation comme production d'un texte dans un langage informatique.

Des éclairages aussi dans le document ressource algorithmique et programmation

1 L'introduction de Python en Seconde

- **Programme de Première :**

Algorithmique et programmation

[…]

La classe de seconde a permis de consolider les acquis du cycle 4 autour de deux idées essentielles :

- la notion de fonction ;
- la programmation comme production d'un texte dans un langage informatique.

[…]

L'accent est mis sur la programmation modulaire qui permet de découper une tâche complexe en tâches plus simples.

1 L’introduction de Python en Seconde

Conclusion :

Écrire des **fonctions** dès que la console est prise en main, en lien avec l’écriture des algorithmes en langage naturel au Baccalauréat.

Plan

1. L’introduction de Python en Seconde
2. **Les types de variables**
3. Une progression en Seconde
4. Les listes
5. Premières séances en Première
6. La trace écrite
7. L’évaluation

2 Les types de variables

- Dans un ordinateur, toutes les données sont codées en **base 2** :

  1 : le courant passe,

  0 : le courant ne passe pas.

- Représentation des **entiers** : type **integer** (int)

  - codés en base 2,

  - avec une limite maximale dépendant de la machine.

- Représentation des **chaînes de caractères** : type **string** (str)
- Représentation des **booléens** : type **boolean** (bool)

2 Les types de variables

- Une **expression booléenne** est une expression qui ne peut prendre que deux valeurs : « vrai » ou « faux ».

```
>>> 7 < 4
False
```

- Une **variable booléenne** est une variable qui ne peut prendre que deux valeurs : « vrai » ou « faux ». C'est le cas de la variable « mineur » ci-dessous :

```
age = 15
mineur = age<18
```

- Les opérateurs de comparaison (==, !=, <, >, ≤, ≥) sont définis en Python pour retourner une valeur booléene.
  Ce résultat peut être exploité immédiatement dans le programme (cas de l'instruction conditionnelle if), ou encore stocké dans une variable pour un usage ultérieur.

2 Les types de variables

- Une fonction peut retourner une valeur booléenne :

```
def multiple1(a, b) :
    if a%b == 0 :
        resultat = True
    else :
        resultat = False
    return resultat
```

- Ou encore :

```
def multiple2(a, b) :
    return a%b == 0
```

2 Les types de variables

- Une fonction peut retourner une valeur booléenne :

```
def multiple1(a, b) :
    if a%b == 0 :
        resultat = True
    else :
        resultat = False
    return resultat
```

variable booléenne

- Ou encore :

```
def multiple2(a, b) :
    return a%b == 0
```

expression booléenne

2 Les types de variables

- Représentation des **entiers** : type **integer** (int),
- Représentation des **chaînes de caractères** : type **string** (str)
- Représentation des **booléens** : type **boolean** (bool)
- Représentation des **décimaux** : nombres à virgule flottante type **float** (float)

2 Les types de variables

- La représentation des **décimaux** par les **nombres à virgule flottante** (type float), est plus problématique :
- En binaire, on utilise une notation analogue à la notation scientifique en base 10 :

$s\ m\ 2^n$, où : s est le signe du nombre,
m est sa mantisse,
n est son exposant.

- Ainsi, seuls les **nombres dyadiques** (de la forme $\frac{A}{2^k}$, A et k entiers) peuvent être représentés de façon exacte.

2 Les types de variables

- Les nombres à virgule flottante ont une précision limitée et fournissent des résultats « surprenants » au premier abord :

```
>>> 0.1 + 0.2
0.30000000000000004
```

- Le test d'égalité « == » est en conséquence mal adapté à la comparaison de deux nombres flottants :

```
>>> 0.1 + 0.2 == 0.3
False
```

2 Les types de variables

- Que penser du programme suivant ?

```
def rectangle(a, b, c) :
    if a**2 + b**2 == c**2 or a**2 + c**2 == b**2 or b**2 + c**2 = a**2 :
        return True
    else :
        return False
```

2 Les types de variables

- **Ressource 3 : AlgoProg_Ressource_3_QCM_logique.pdf**

Question 2 :

```
1 4 < 5
```

☐ True ☐ False ☐ Erreur ☐ 4 < 5

Question 3 :

```
1 3 * 4 < 3 * 5
```

☐ True ☐ False ☐ Erreur ☐ 3 * 4 < 3 * 5

Question 4 :

```
1 -3 * 4 < -3 * 5
```

☐ True ☐ False ☐ Erreur ☐ -3 * 4 < -3 * 5

2 Les types de variables

- **Ressource 3 : AlgoProg_Ressource_3_QCM_logique.pdf**

Question 8 :

```
1 not (4 > 7)
```

☐ True ☐ False ☐ Erreur ☐ not 4 > 7

Question 9 :

```
1 not True
```

☐ True ☐ False ☐ Erreur ☐ not True

Question 10 :

```
1 not False
```

☐ True ☐ False ☐ Erreur ☐ not False

2 Les types de variables

- **Ressource 3 : AlgoProg_Ressource_3_QCM_logique.pdf**

Question 11 :

```
1 (3 < 5) and (4 > 7)
```

☐ True ☐ False ☐ Erreur ☐ 3 < 5 and 4 > 7

Question 12 :

```
1 True and 3 > 5
```

☐ True ☐ False ☐ Erreur ☐ not 3 < 5

Question 13 :

```
1 True and False
```

☐ True ☐ False ☐ Erreur ☐ not 4 > 7

Plan

1. L’introduction de Python en Seconde
2. Les types de variables
3. **Une progression en Seconde**
4. Les listes
5. Premières séances en Première
6. La trace écrite
7. L’évaluation

3 Une progression en Seconde

- Un **premier T.P.** envisageable : **Console, variables et algorithmes.**

- Exemple :

Algorithme	**Python**	**Langage naturel**
x reçoit la valeur 2	`x = 2`	$x \leftarrow 2$
x reçoit la valeur $x+3$	`x = x + 3`	$x \leftarrow x+3$

3 Une progression en Seconde

- Un **second T.P. : fonctions informatiques**

- Exemples :

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

```
def triangle(base, hauteur) :
    aire = ( ..... * ..... ) / 2
    return .....
```

3 Une progression en Seconde

- Un **troisième T.P.** : **L'instruction conditionnelle**

- Exemple :

```
def prix(age) :
    """ Retourne le tarif d'entrée au cinéma selon l'age """
    if age < 18 :
        tarif = 4
    else :
        tarif = 7
    return tarif
```

3 Une progression en Seconde

- Un **quatrième T.P.** : **la boucle while** et applications aux fonctions

- Exemple, en fin de chapitre :

Exercice : modéliser une propagation épidémique

- considérons une population (humaine, animale …),
- au jour 0, trois individus sont contaminés par le virus d'une épidémie,
- chaque jour, le nombre d'individus contaminés triple.

a. Écrire une fonction Python, prenant en argument l'effectif de la population, et renvoyant le premier jour où toute la population sera contaminée.

b. Utiliser la fonction pour déterminer au bout de combien de jour une population d'un million d'individus sera entièrement contaminée.

c. Notre modèle (→ triple tous les jours), est très simplifié. Mais il a déjà une caractéristique très nette : ce modèle vous semble-t-il correspondre à une épidémie se propageant rapidement ou lentement ?

d. Proposer en langage naturel un algorithme correspondant à une épidémie où le nombre d'individus contaminés augmente de 5% tous les jours.

3 Une progression en Seconde

- Un **cinquième T.P.** : **Types et Arithmétique**

- Déterminer si un entier naturel *a* est multiple d'un entier naturel *b* :

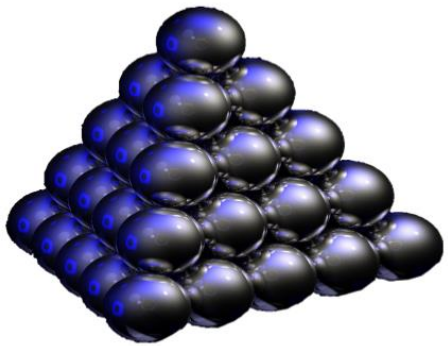
```
def multiple(a, b) :
    """ Determine si a est multiple de b """
    if a % b == 0 :
        resultat = True
    else :
        resultat = False
    return resultat
```

3 Une progression en Seconde

- Un **sixième T.P.** : la **boucle for … in …**

Exercice : un problème d'empilement de sphères

On empile des sphères comme sur la figure ci-dessous pour former une pyramide à base carrée.



a. Combien de sphères faut-il pour réaliser une pyramide à 2 étages ? 3 étages ?

b. Écrire une fonction ''pyramide(n) '' renvoyant le nombre de sphères nécessaires pour réaliser une pyramide à n étages.

3 Une progression en Seconde

- Un **septième T.P.** : Échantillonnage

- Exemple :

```
from random import randint

def frequence_succes(n) :
    """ Renvoie la fréquence de 6 obtenus sur n lancers """
    succes = 0
    for compteur in range(n) :
        de = randint(1, 6)
        if de == 6 :
            succes = succes + 1
    return succes / n
```

- Observation lorsque n devient « grand ».

3 Une progression en Seconde

- **Exemples d'objectifs de fin d'année :**

a) Algorithme d'approximation d'un extremum de fonctions

b) Déterminer par balayage un encadrement de $\sqrt{2}$

c) Déterminer si un nombre est premier

```
def multiple(a, b) :
    """ Renvoie si a est multiple de b """
    if a % b == 0:
        resultat = True
    else :
        resultat = False
    return resultat

def premier(n) :
    """ Determine si le nombre n est premier """
    resultat = True
    for compteur in range(2, n) :
        if multiple(n, compteur) :
            resultat = False
    return resultat
```

Plan

1. L’introduction de Python en Seconde
2. Les types de variables
3. Une progression en Seconde
4. **Les listes**
5. Premières séances en Première
6. La trace écrite
7. L’évaluation

4 Les listes

- Une liste est une collection ordonnée d'objets :

```
liste_1 = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]

liste_2 = ["Janvier", "Février", "Mars", "Avril", "Mai", "Juin"]
```

- Une première application en mathématiques : créer la liste des premiers termes d'une suite :

```
premiers = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
```

- Le caractère « ordonné » différencie les listes des ensembles :

```
[1, 2] != [2, 1]
[1, 2, 1] != [1, 2]
```

Créer une liste en extension

- En python, les listes sont des objets de type « list ».
- On peut générer une liste « en extension », en la saisissant manuellement, entre crochets, en séparant les éléments pas des virgules :

```
premiers = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
```

Accéder aux éléments d'une liste

- Les éléments d'une liste sont numérotés à partir de 0 (de même que u_0 est souvent le premier terme d'une suite).
- On accède à un élément de la liste en indiquant l'indice entre crochets :

```
>>> liste = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> liste[0]
>>> 1

>>> liste = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> liste[3]
>>> 4

>>> liste = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> liste[3:5]
>>> [3, 4]
```

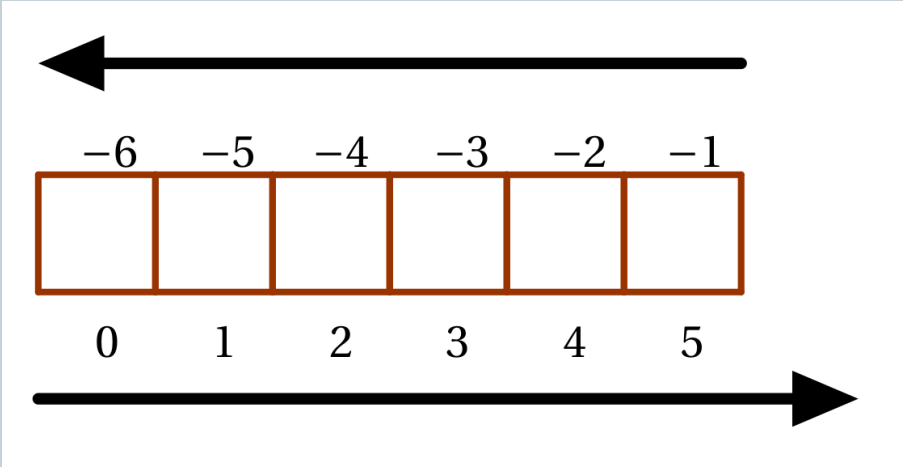
Accéder aux éléments d'une liste

- Il est parfois utile de spécifier un indice en partant de la fin de la liste :

```
>>> liste = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> liste[-1]
10

>>> liste = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> liste[-2]
9
```

- En résumé :



Générer une liste par ajouts successifs

- Lorsque la liste s'allonge, Python offre des mécanismes de création des listes plus adaptés que la saisie en extension.
- Générer une liste par ajouts successifs :
  une fois la liste créée, la méthode **append** permet de la compléter par **ajouts successifs** :

```
1# 1) Création d'une liste vide :
2liste = []
3
4# 2) Ajouts successifs des 100 premiers entiers :
5for compteur in range(1, 101) :
6    liste.append(compteur)
```

Générer une liste en compréhension

```
>>> liste = [n for n in range(1, 101)]

>>> liste
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
 12, 13, 14, 15, 16, 17, 18, 19, 20,
 21, 22, 23, 24, 25, 26, 27, 28, 29,
 30, 31, 32, 33, 34, 35, 36, 37, 38,
 39, 40, 41, 42, 43, 44, 45, 46, 47,
 48, 49, 50, 51, 52, 53, 54, 55, 56,
 57, 58, 59, 60, 61, 62, 63, 64, 65,
 66, 67, 68, 69, 70, 71, 72, 73, 74,
 75, 76, 77, 78, 79, 80, 81, 82, 83,
 84, 85, 86, 87, 88, 89, 90, 91, 92,
 93, 94, 95, 96, 97, 98, 99, 100]
```

Générer une liste en compréhension

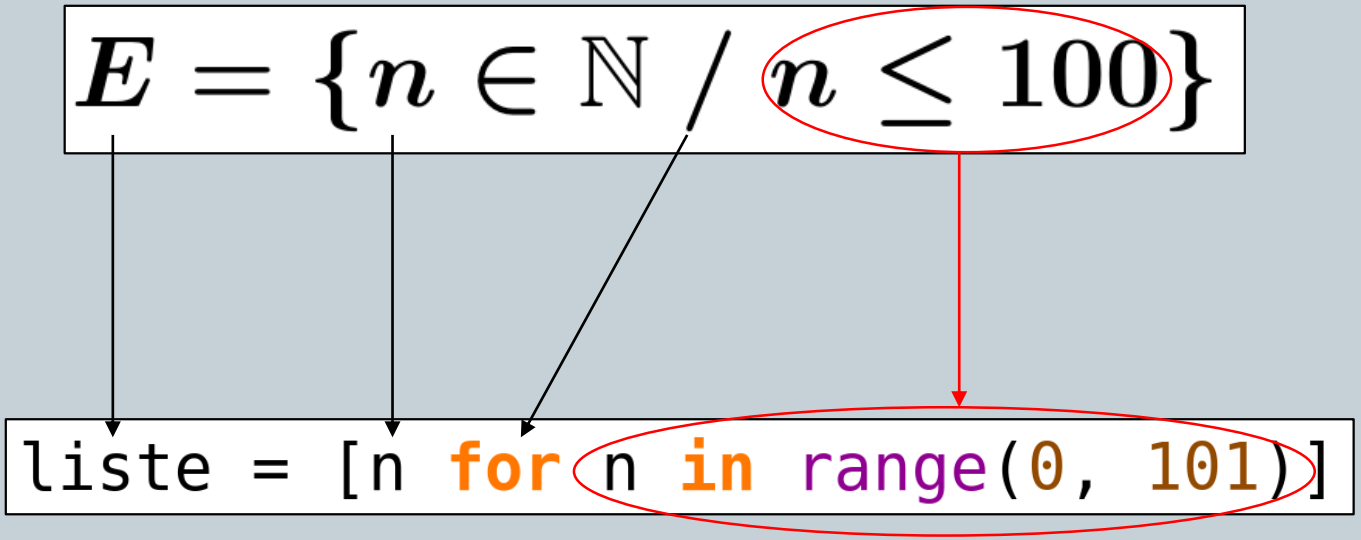
- Une méthode qui se rapproche de la notation mathématique des ensembles :

$$E = \{n \in \mathbb{N} \,/\, n \leq 100\}$$

```
liste = [n for n in range(0, 101)]
```

Générer une liste en compréhension

- Une méthode qui se rapproche de la notation mathématique des ensembles :



- Remarque : on peut aussi générer des ensembles en compréhension.

Générer une liste en compréhension

- Générer une liste de multiples de 3 :

```
>>> triple = [3*i for i in range(1,  10)]
>>> triple
[3, 6, 9, 12, 15, 18, 21, 24, 27]
```

- Générer une liste de puissance de 2 :

```
>>> puissance_2 = [2**i for i in range(1, 10)]
>>> puissance_2
[2, 4, 8, 16, 32, 64, 128, 256, 512]
```

Générer une liste en compréhension

- Il est aussi possible d'introduire une condition avec un « if » :

```
>>> pas_multiple_de_7 = [i for i in range(1, 101) if i%7 != 0]

>>> pas_multiple_de_7
[1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 22,
 23, 24, 25, 26, 27, 29, 30, 31, 32, 33, 34, 36, 37, 38, 39, 40, 41,
 43, 44, 45, 46, 47, 48, 50, 51, 52, 53, 54, 55, 57, 58, 59, 60, 61,
 62, 64, 65, 66, 67, 68, 69, 71, 72, 73, 74, 75, 76, 78, 79, 80, 81,
 82, 83, 85, 86, 87, 88, 89, 90, 92, 93, 94, 95, 96, 97, 99, 100]
```

Manipuler les éléments d'une liste

- Les fonctions permettant de manipuler un type d'objet particulier s'appellent des **méthodes**.

- Elles s'utilisent selon la **syntaxe** :

  - objet.methode(*arguments*)

Manipuler les éléments d'une liste

- La méthode *append* permet d'ajouter un élément en fin de liste.
- La méthode *pop* permet de supprimer un élément en spécifiant son indice :

```
>>> liste = [0, 1, 2, 3, 4, 5]
>>> liste.pop(2)
>>> 2
>>> liste
>>> [0, 1, 3, 4, 5]
```

Manipuler les éléments d'une liste

- Il existe de très nombreuses autres **méthodes** sur les listes :

  - La méthode *remove* : supprime la première occurrence d'une valeur,

  - La méthode *insert* : permet d'insérer un élément à un indice à spécifier,

  - La méthode *count* : renvoie le nombre d'occurrences d'une valeur,

  - La méthode *sort* : permet de ranger une liste par ordre croissant.

- Enfin certaines **fonctions natives** de Python peuvent être utilisées avec une liste en argument :

  - len(liste) : renvoie la longueur de liste,

  - max(liste) : renvoie le maximum de liste, si celui-ci a un sens.

Parcourir une liste

- Deux principaux **modes de parcours des listes** sont disponibles :
- **a. Parcourir une liste à l'aide des indices :**

```
>>> liste = [0, 10, 20, 30, 40, 50]
>>> for compteur in range(0, 6) :
...     print(liste[compteur])
...
0
10
20
30
40
50
```

La variable *compteur* prend successivement les différentes valeurs d'indices de 0 à 5.

Itérer sur les éléments d'une liste

- **b. parcourir une liste par ses éléments :**

```
>>> semaine = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi']
>>> for jour in semaine :
...     print(jour)
...
lundi
mardi
mercredi
jeudi
vendredi
```

La variable *jour* prend successivement les différentes valeurs des éléments de la liste *semaine*.

QCM sur les listes

- **Un QCM sur les listes :**
- **Ressource 4 : AlgoProg_Ressource_4_QCM_listes.pdf**

Plan

1. L’introduction de Python en Seconde
2. Les types de variables
3. Une progression en Seconde
4. Les listes
5. **Premières séances en Première**
6. La trace écrite
7. L’évaluation

5 Premières séances en Première

Que faire lors de la première séance informatique de l'année ?

5 Premières séances en Première

- **Des possibilités :**

  1) Un **TP-0 différencié** avec l'aide d'un **glossaire de Seconde** :

  - **réactivation** de Python avec des exercices types de Seconde
  - **approfondissements** pour les élèves plus avancés (suivant la spécialité N.S.I. par exemple)

  2) Des exercices d'**automatismes**, pour réactiver :
  - les différentes structures de programmation
  - les syntaxes Python

5 Premières séances en Première

Concernant les listes :

- Quelle entrée sur les listes en première ?

- Sur quelles notions mathématiques s'appuyer ?

5 Une introduction des listes en Première

Ressource_5 : AlgoProg_Ressource5_Premiere_TP1.pdf

- **Points d'appuis :**

  - les suites
  - ensembles mathématiques
  - notions de Seconde en algorithmique et programmation (un glossaire de seconde peut être proposé)

- **Objectifs :**

  - création de listes en extension
  - accès aux éléments d'une liste
  - création de listes par ajouts successifs
  - liste des premiers termes d'une suite définie par récurrence

- **Modalités :**

  - en salle informatique
  - une heure

- **Prolongement du T.P. en devoir à la maison :**

  - la suite de Fibonacci (cf 7. Évaluation)

Plan

1. L’introduction de Python en Seconde
2. Les types de variables
3. Une progression en Seconde
4. Les listes
5. Premières séances en Première
6. **La trace écrite**
7. L’évaluation

6 La trace écrite

- Elle doit permettre de :

  - synthétiser les notions étudiées,

  - offrir un document de référence où l'élève pourra retrouver une définition, une méthode, ou une syntaxe ...

- Exemples :

Ressource 6 : AlgoProg_Ressource_6_TraceEcrite_TP1&2.pdf

Ressource 7 : AlgoProg_Ressource_7_TraceEcrite_for.pdf

Plan

1. L’introduction de Python en Seconde
2. Les types de variables
3. Une progression en Seconde
4. Les listes
5. Premières séances en Première
6. La trace écrite
7. **L’évaluation**

7 L’évaluation

En salle informatique :

- à l’aide d’une grille d’évaluation par compétences explicites,
- pour tous les élèves ou seulement pour certains,
- une opportunité de valoriser des compétences telles que la curiosité, l’esprit d’initiative, la ténacité.

7 L'évaluation

- **Exemple de séance** :

- **Première partie :**

- découverte d'une approximation historique de π par $\frac{22}{7}$
- obtenir une approximation de π dans Python
- erreur lors de l'approximation de π par $\frac{22}{7}$

- **Seconde partie :**

Écrire un algorithme permettant de trouver les fractions dont le numérateur et le dénominateur sont des entiers à 1, 2 ou 3 chiffres, et qui fournissent une meilleure approximation de π que $\frac{22}{7}$?

7 L'évaluation

Trouver des fractions dont le numérateur et le dénominateur sont des entiers à 1, 2 ou 3 chiffres qui fournissent une meilleure approximation de π que $\frac{22}{7}$.

Raisonner (analyser, décomposer un problème ; corriger, optimiser un algorithme)	- L'élève témoigne au moins à l'oral de sa compréhension de l'énoncé - Des corrections successives sont apportées au script au vu des affichages obtenus
Modéliser (mettre en œuvre un algorithme correspondant à un problème donné ; élaborer une simulation numérique)	- L'inventaire de toutes les fractions est modélisée par 2 boucles for imbriquées - La notion de meilleure approximation est associée au calcul de l'erreur ou à un encadrement
Produire un texte dans un langage informatique (programmer une affectation, une boucle, une instruction conditionnelle)	- L'affectation est traduite par une égalité - L'indentation est respectée - La fonction range est correctement utilisée (décalage pris en compte)
Concevoir un algorithme correspondant à un problème donné (concevoir une affectation; programmer une boucle, une instruction conditionnelle)	- Une variable de type erreur = pi – a/b est introduite dans le script - Le problème du signe de l'erreur est pris en compte pour envisager une disjonction de cas

7 L'évaluation

- Des possibilités en **devoir à la maison** :

  - réinvestir les acquis d'un T.P.

  - résolution d'un problème :
    traiter à l'aide de l'informatique un problème de mathématiques

- Exemples :

Ressource 8 : AlgoProg_Ressource_8_DM_Seconde.pdf

Ressource 9 : AlgoProg_Ressource_9_DM_Fibonacci.pdf

7 L'évaluation

- **En contrôle** :
  (a) exercices sur les **automatismes** :

Exercice :
Pour chacune des séquences d'instructions, prévoir ce que contiendra la variable écrite en dernière ligne.

`>>> a = 4`
`>>> a = a + 3`
	`>>> age = 15`
	`>>> age = age + 1`
		`>>> prix_ht = 20`
		`>>> tva = 1.2`
		`>>> prix_ttc = prix_ht * tva`

`>>> b = 5`
`>>> b = b + 3`
`>>> b = 2 * b`
	`>>> b = 5`
	`>>> b = 2 * b`
	`>>> b = b + 3`
		`>>> a = 17`
		`>>> b = 33`
		`>>> somme = a + b`

		`>>> nombre = 12` `>>> resultat = nombre ** 2`

7 L'évaluation

- **En contrôle** :
  (a) exercices sur les **automatismes** :

```
1 multiple = []
2 for compteur in range(0, 5) :
3     multiple.append(3*compteur)
```

Question 14 : Que contient la liste `multiple` ?

☐ [0,3,6,9,12,15] ☐ [3,6,9,12,15] ☐ [0,3,6,9,12] ☐ [0,3,6,9,12,15,18]

```
1 entiers = [i for i in range(0, 100)]
```

Question 15 : Que contient la liste `entiers` ?

☐ entiers de 0 à 100 ☐ 0 et 100 ☐ entiers de 1 à 99 ☐ entiers de 0 à 99

```
1 multiple2 = [4*i for i in range(0, 6)]
```

Question 16 : Que contient la liste `multiple2` ?

☐ [4,8,12,16,20,24] ☐ [4,8,12,16,20] ☐ [0,4,8,12,16,20] ☐ [0,4,8,12,16,20,24]

7 L'évaluation

- **En contrôle** :
  (a) exercices sur les **automatismes** :

Exercice :
Pour chacun des algorithmes : prévoir ce que contiendra la variable écrite en dernière ligne.

$b \leftarrow 7$ $b \leftarrow 4\,b$	$annee \leftarrow 2018$ $annee \leftarrow annee + 1$	$mise \leftarrow 10$ $mise \leftarrow mise \times 1{,}1$
$x \leftarrow 12$ $x \leftarrow 5 \times x$ $x \leftarrow x - 12$	$x \leftarrow 4$ $y \leftarrow 7$ $resultat \leftarrow 2x \times y$	$a \leftarrow 3$ $b \leftarrow 2$ $difference\_carre \leftarrow (a+b)(a-b)$

7 L'évaluation

- **En contrôle** :
- (b) **compléter** un programme :

```
def triangle(base, hauteur) :
    aire = ( ..... * ..... ) / 2
    return .....
```

7 L'évaluation

- **En contrôle** :
  (b) **compléter** un programme :

Exercice 1 : espérance et variance d'une variable aléatoire

On donne ci-dessous une fonction permettant de calculer l'espérance d'une variable aléatoire :

```
1  def esperance(valeurs, probabilites) :
2      """ Calcule l'espérance d'une variable aléatoire """
3      resultat = 0
4      for compteur in range(0, len(valeurs)) :
5          resultat = resultat + valeurs[compteur] * probabilites[compteur]
6      return resultat
```

Proposer une fonction "variance" renvoyant la variance d'une variable aléatoire :

```
1  def variance(valeurs, probabilites) :
2      """ Calcule la variance d'une variable aléatoire """
3      ..........................................................................
4      ..........................................................................
5      ..........................................................................
6      ..........................................................................
7      ..........................................................................
```

7 L'évaluation

- **En contrôle** :

(c) **comprendre** un programme et **prévoir** le résultat :

Exercice :
On donne le programme suivant :

```
1  def calcul(n) :
2      n = n + 2
3      n = 3 * n
4      return n
```

a. Prévoir le résultat de l'instruction : `>>> calcul(5)`.

b. Écrire l'algorithme correspondant en langage naturel.

7 L'évaluation

- **En contrôle :**

  (c) **comprendre** un programme et **l'écrire en langage naturel**

Exercice :
On donne le programme suivant :

```
def epidemie(n) :
    malade = 1000
    jour = 0
    while malade <= 10 000 :
        jour = jour + 1
        malade = malade * 1.1
    return jour
```

a. Quel est le rôle de cette fonction.

b. Utiliser la calculatrice pour déterminer le résultat renvoyé par l'instruction : `>>> epidemie(2000)`.

c. Écrire cette fonction en langage naturel.