Chapitres

plan

Une première réponse souvent avancée...

L'initiation à l'informatique répond au besoin d'un éveil au phénomène socio-culturel et technologique que constitue l'informatique et la révolution numérique.

...Qui cache les véritables enjeux

Quelques objectifs

...Qui cache les véritables enjeux

Quelques objectifs

- Capacité à poser un problème

...Qui cache les véritables enjeux

Quelques objectifs

- Capacité à poser un problème
- Décomposer un problème en sous problèmes

...Qui cache les véritables enjeux

Quelques objectifs

- Capacité à poser un problème
- Décomposer un problème en sous problèmes
- Développement de la pensée logique

...Qui cache les véritables enjeux

Quelques objectifs

- Capacité à poser un problème
- Décomposer un problème en sous problèmes
- Développement de la pensée logique
- Planifier des actions

...Qui cache les véritables enjeux

Quelques objectifs

- Capacité à poser un problème
- Décomposer un problème en sous problèmes
- Développement de la pensée logique
- Planifier des actions
- Approche par essai erreur "debugging"

...Qui cache les véritables enjeux

Quelques objectifs

- Capacité à poser un problème
- Décomposer un problème en sous problèmes
- Développement de la pensée logique
- Planifier des actions
- Approche par essai erreur "debugging"
- Anticiper des résultats

...Qui cache les véritables enjeux

Quelques objectifs

- Capacité à poser un problème
- Décomposer un problème en sous problèmes
- Développement de la pensée logique
- Planifier des actions
- Approche par essai erreur "debugging"
- Anticiper des résultats
- Développement de la rigueur de la pensée et précision de l'expression

...Qui cache les véritables enjeux

Quelques objectifs

- Capacité à poser un problème
- Décomposer un problème en sous problèmes
- Développement de la pensée logique
- Planifier des actions
- Approche par essai erreur "debugging"
- Anticiper des résultats
- Développement de la rigueur de la pensée et précision de l'expression
- Compréhension de concepts généraux (variables, fonctions)

plan

simplification de la syntaxe, avec le symbole « $\longleftarrow$ » pour l'affectation.

simplification de la syntaxe, avec le symbole « $\longleftarrow$ » pour l'affectation.

- Une affection se lit de la droite vers la gauche : on calcule en premier, on affecte en deuxième

simplification de la syntaxe, avec le symbole « $\longleftarrow$ » pour l'affectation.

- Une affection se lit de la droite vers la gauche : on calcule en premier, on affecte en deuxième
- Ne pas le confondre avec le = mathématiques

simplification de la syntaxe, avec le symbole « $\longleftarrow$ » pour l'affectation.

- Une affection se lit de la droite vers la gauche : on calcule en premier, on affecte en deuxième
- Ne pas le confondre avec le = mathématiques
- L'affectation est le concept algorithmique le plus dire à concevoir car pas naturel

suppression des entrées-sorties

suppression des entrées-sorties

- Utilisation de l'interpréteur

suppression des entrées-sorties

- Utilisation de l'interpréteur
- Plus de print et de input : destructeur

suppression des entrées-sorties

- Utilisation de l'interpréteur
- Plus de print et de input : destructeur
- Elles apparaissent naturellement dans les fonctions

suppression des entrées-sorties

- Utilisation de l'interpréteur
- Plus de print et de input : destructeur
- Elles apparaissent naturellement dans les fonctions
- print différent du return

suppression des entrées-sorties

- Utilisation de l'interpréteur
- Plus de print et de input : destructeur
- Elles apparaissent naturellement dans les fonctions
- print différent du return
- print pour le déboggage

suppression de la déclaration des variables, les hypothèses faites sur les variables étant précisées par ailleurs

suppression de la déclaration des variables, les hypothèses faites sur les variables étant précisées par ailleurs

- Précisions dans l'énonce

suppression de la déclaration des variables, les hypothèses faites sur les variables étant précisées par ailleurs

- Précisions dans l'énonce
- Commentaire dans le programme

suppression de la déclaration des variables, les hypothèses faites sur les variables étant précisées par ailleurs

- Précisions dans l'énonce
- Commentaire dans le programme
- plus avancé : utilisation de assert

fin du recours aux tests d'égalité entre flottants

fin du recours aux tests d'égalité entre flottants

- 0.1 + 0.2 ; 3.11+2.08

fin du recours aux tests d'égalité entre flottants

- 0.1 + 0.2 ; 3.11+2.08
- renoncer à présenter des programmes qui y font appel (Pythagore)

$$\sqrt{2} * *2 + \sqrt{3} * *2 == \sqrt{5} * *2$$

fin du recours aux tests d'égalité entre flottants

- 0.1 + 0.2 ; 3.11+2.08
- renoncer à présenter des programmes qui y font appel (Pythagore)

$$\sqrt{2} * *2 + \sqrt{3} * *2 == \sqrt{5} * *2$$

- 28 morts pendant la guerre du golf

Un usage plus raisonné des booléens qui peuvent fort bien faire l'objet d'une affectation.

Un usage plus raisonné des booléens qui peuvent fort bien faire l'objet d'une affectation.

- Exemple

Un usage plus raisonné des booléens qui peuvent fort bien faire l'objet d'une affectation.

- Exemple

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

res = est_divisible(1243,97)
```

```
res = 1243678637487 4%97 == 0
```

Un usage plus raisonné des booléens qui peuvent fort bien faire l'objet d'une affectation.

- Exemple

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

res = est_divisible(1243,97)
```

```
res = 1243678637487 4%97 == 0
```

- Cette dernière écriture est moins explicite et n'est pas à utiliser en premier abord : c'est un raccourci à construire avec les élèves

Les fonctions sont à voir dès le mois de septembre de seconde.

Les fonctions sont à voir dès le mois de septembre de seconde.

- Pourquoi s'en passer

Les fonctions sont à voir dès le mois de septembre de seconde.

- Pourquoi s'en passer
- Les fonctions informatiques (def) SONT des fonctions mathématiques

Les fonctions sont à voir dès le mois de septembre de seconde.

- Pourquoi s'en passer
- Les fonctions informatiques (def) SONT des fonctions mathématiques
- Les entrées-sorties sont explicites

Les fonctions sont à voir dès le mois de septembre de seconde.

- Pourquoi s'en passer
- Les fonctions informatiques (def) SONT des fonctions mathématiques
- Les entrées-sorties sont explicites
- Construit la notion de paramètres/variables

Les fonctions sont à voir dès le mois de septembre de seconde.

- Pourquoi s'en passer
- Les fonctions informatiques (def) SONT des fonctions mathématiques
- Les entrées-sorties sont explicites
- Construit la notion de paramètres/variables
- Que des avantages

Pas de cours chapitré sur l'algorithme

l'usage majoritaire de thèmes liés aux contenus du programme de mathématiques.

Pas de fonctions à images multiples.

Pas de fonctions à images multiples.

- L'image est en fait unique : tuple

Pas de fonctions à images multiples.

- L'image est en fait unique : tuple
- Paramètres multiples : oui

Exemple : montecarlo

```
from random import *
from math import *
compteur = 0
for k in range(1000000) :
    x = 4*random()-2
    y = random()
    if y<exp(-x**2/2) :
        compteur = compteur + 1
frequence = compteur /1000000
aire = 4*frequence
```

Exemple : montecarlo avec une fonction

Si l'on veut la fréquence :

```
from random import *
from math import *
def montecarlo(n) :
    compteur = 0
    for k in range(n) :
        x = 4*random()-2
        y = random()
        if y<exp(-x**2/2) :
            compteur =
                compteur + 1
    frequence = compteur /n
    aire = 4*frequence
    return frequence
```

Si l'on veut l'aire :

```
from random import *
from math import *
def montecarlo(n) :
    compteur = 0
    for k in range(n) :
        x = 4*random()-2
        y = random()
        if y<exp(-x**2/2) :
            compteur =
                compteur + 1
    frequence = compteur /n
    aire = 4*frequence
    return aire
```

A proscrire :

```
from random import *
from math import *
def montecarlo(n) :
    compteur = 0
    for k in range(n) :
        x = 4*random()-2
        y = random()
        if y<exp(-x**2/2) :
            compteur = compteur + 1
    frequence = compteur /n
    aire = 4*frequence
    return aire,frequence
```

A plusieurs paramètres :

```
from random import *
from math import *
def montecarlo(n,a,b,f) :
    """hypothese : f positive de maximum inferieur a 1"""
    compteur = 0
    for k in range(n) :
        x = (b-a)*random()+a
        y = random()
        if y<f(x) :
            compteur = compteur + 1
    frequence = compteur /n
    aire = (b-a)*frequence
    return aire
```

plan

instructions de bases

```
ma_liste = [] #liste vide
ma_liste2 = [2,"bonjour",3.14,montecarlo]

ma_liste2[2] #renvoie le 2eme element de ma_liste  : 3.14
ma_liste2[0] #renvoie 2

ma_liste.append(7) #Ajoute 7 a ma_liste
ma_liste = ma_liste + [7] #A proscrire

fusion = ma_liste + ma_liste2 #concatenation
```

Applications à Syracuse

Fonction qui renvoie le suivant de la trajectoire de Syracuse

```
def suivant_syracuse(nombre) :
    if nombre%2 == 0 :
        return nombre//2
    else :
        return 3*nombre + 1
```

Fonction qui renvoie le temps de vol d'un nombre dans la trajectoire de Syracuse

```
def temps_vol(nombre) :
    compteur = 0
    while nombre  != 1 :
        nombre = suivant_syracuse(nombre) #tres difficile
        compteur = compteur + 1
    return compteur
```

Applications à Syracuse

Génération d'une liste par ajout successif

Applications à Syracuse

Génération d'une liste par ajout successif

```
temps = [0]
for k in range(1,1000) :
    temps.append(temps_vol(k))
```

Applications à Syracuse

Génération d'une liste par ajout successif

```
temps = [0]
for k in range(1,1000) :
    temps.append(temps_vol(k))
```

- temps contiendra tous les temps de vols des nombres de 0 à 1000

Applications à Syracuse

Génération d'une liste par ajout successif

```
temps = [0]
for k in range(1,1000) :
    temps.append(temps_vol(k))
```

- temps contiendra tous les temps de vols des nombres de 0 à 1000
- temps[14] est le temps de vol de 14

Applications à Syracuse

Génération d'une liste par compréhension

Applications à Syracuse

Génération d'une liste par compréhension

```
temps = [temps_vol(nombre) for nombre in range(1,1000)]
```

Applications à Syracuse

Génération d'une liste par compréhension

- ```
  temps = [temps_vol(nombre) for nombre in range(1,1000)]
  ```
- temps contiendra tous les temps de vols des nombres de 1 à 1000
  ```

Applications à Syracuse

Génération d'une liste par compréhension

```
temps = [temps_vol(nombre) for nombre in range(1,1000)]
```

- temps contiendra tous les temps de vols des nombres de 1 à 1000
- temps[13] est le temps de vol de 14

Autres intérêts

Autres intérêts

Autres intérêts

Autres intérêts

```
for machin in liste :
    traitement(machin)
```

Autres intérêts

Autres intérêts

- ```
  for machin in liste :
      traitement(machin)
  ```
- possibilité de travailler avec des listes sans s'occuper du rang
  ```

Autres intérêts

exemple

Autres intérêts

exemple

```
nombres = [1000*random() for k in range(10000)]
```

Autres intérêts

exemple

```
nombres = [1000*random() for k in range(10000)]
```

- nombres est une liste de 10 000 nombres entre 0 et 1000.

Autres intérêts

exemple

```
nombres = [1000*random() for k in range(10000)]
```

- nombres est une liste de 10 000 nombres entre 0 et 1000.
- Comment extraire ceux qui sont plus petit que 400 ?

Autres intérêts

exemple

```
nombres = [1000*random() for k in range(10000)]
```

- nombres est une liste de 10 000 nombres entre 0 et 1000.
- Comment extraire ceux qui sont plus petit que 400 ?
- par ajouts successifs :

```
nombres400 = []
for nombre in nombres :
    if nombre<400 :
        nombres400.append(nombre)
```

Autres intérêts

exemple

```
nombres = [1000*random() for k in range(10000)]
```

- nombres est une liste de 10 000 nombres entre 0 et 1000.
- Comment extraire ceux qui sont plus petit que 400 ?
- par ajouts successifs :

```
nombres400 = []
for nombre in nombres :
    if nombre<400 :
        nombres400.append(nombre)
```

- par compréhension :

```
nombres400 = [nombre for nombre in nombres if nombre<400]
```

Autres intérêts

Traitements d'une liste par une fonction

Autres intérêts

Traitements d'une liste par une fonction

```
n_liste = [traitement(machin) for machin in liste if
    condition]
```