Programmation impérative

Fabrice Lembrez - PSI*

Lycée Pierre de Fermat

Algorithmique pure ; il ne sera pas particulièrement question du langage Python.
Chapitre I du poly, sections I et II.

Plan

Les paradigmes de programmation

Déf - Paradigme de programmation

Mode selon lequel on envisage l'écriture de programmes.

Deux paradigmes dont vous avez l'habitude

Programmation impérative

- une succession d'instructions : le **flux**
- des **structures de contrôle** du flux (if/for/...)
- on dit que c'est de la programmation structurée si le code est découpé en "modules" hiérarchisés.

Les paradigmes de programmation

Déf - Paradigme de programmation

Mode selon lequel on envisage l'écriture de programmes.

Deux paradigmes dont vous avez l'habitude

Programmation impérative

- une succession d'instructions : le **flux**
- des **structures de contrôle** du flux (if/for/...)
- on dit que c'est de la programmation structurée si le code est découpé en "modules" hiérarchisés.

Programmation fonctionnelle

- le concept de base est celui de fonction
- la composition remplace les structures de contrôle.

Les paradigmes de programmation

Déf - Paradigme de programmation

Mode selon lequel on envisage l'écriture de programmes.

Deux paradigmes dont vous avez l'habitude

Programmation impérative

- une succession d'instructions : le **flux**
- des **structures de contrôle** du flux (if/for/...)
- on dit que c'est de la programmation structurée si le code est découpé en "modules" hiérarchisés.

Programmation fonctionnelle

- le concept de base est celui de fonction
- la composition remplace les structures de contrôle.

Votre approche de la logique de la programmation fonctionnelle : la récursivité, qui demande de "penser autrement".

Deux autres paradigmes utiles

Parce qu'on les rencontre couramment

Programmation orientée objet rend autonomes des "briques" appelées objets, portant leurs propres variables et fonctions :

En Python tout est objet, d'où certaines syntaxes

Valeur d'un attribut = une variable

`machin.valeur`

Appliquer une méthode = une fonction

`machin.action(paramètres)`

Deux autres paradigmes utiles

Parce qu'on les rencontre couramment

Programmation orientée objet rend autonomes des "briques" appelées objets, portant leurs propres variables et fonctions :

En Python tout est objet, d'où certaines syntaxes

Valeur d'un attribut = une variable

`machin.valeur`

Appliquer une méthode = une fonction

`machin.action(paramètres)`

programmation événementielle : on programme la réaction à des événements (clic de souris, horloge...)

Deux autres paradigmes utiles

Parce qu'on les rencontre couramment

Programmation orientée objet rend autonomes des "briques" appelées objets, portant leurs propres variables et fonctions :

En Python tout est objet, d'où certaines syntaxes

Valeur d'un attribut = une variable

`machin.valeur`

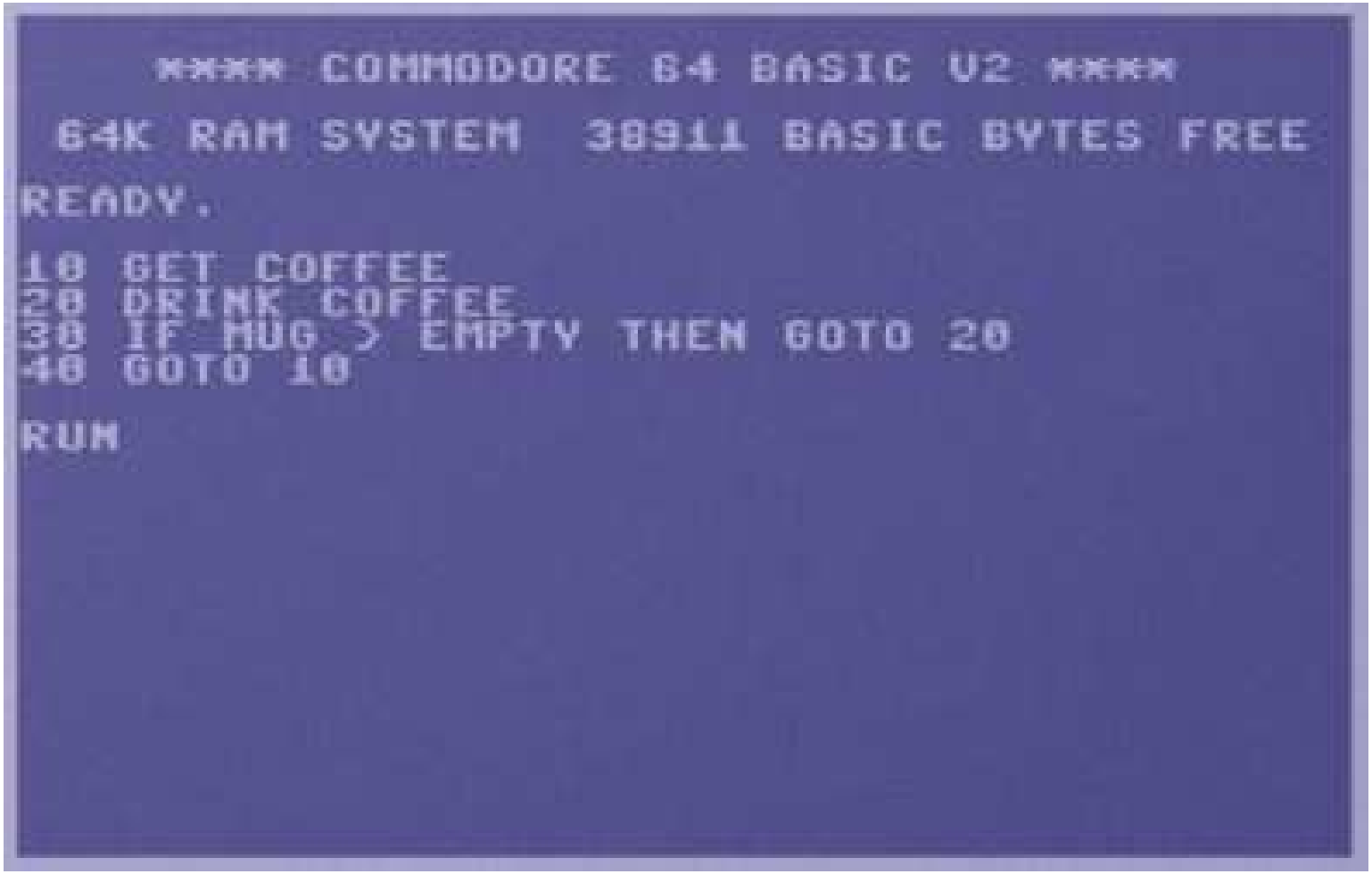
Appliquer une méthode = une fonction

`machin.action(paramètres)`

programmation événementielle : on programme la réaction à des événements (clic de souris, horloge...)

Les langages sont souvent conçus pour utiliser plusieurs paradigmes, et en privilégier certains. Par exemple : POO en Python, programmation fonctionnelle en Caml.

Programmation impérative non structurée



Programme bidon, cependant on numérotait vraiment les lignes et on utilisait GOTO (branchement pur) qui est interdit en programmation structurée.

Programmation impérative non structurée

1/6 du listing d'un jeu paru en 1984 ("Trapped")

```
0 V=53248:S=54272:POKE54296,15:FORI=STOS+24:POKEI,0:NEXT:GOTO27
1 V=53248:POKEV+28,2:POKE53275,2:JV=PEEK(56320):IFSX=0THENSX=5
2 POKEV+28,2:POKEV+37,8:POKEV+38,4:FR=JVANDI6:REM FIRE BUTTON STATUS
3 JV=15-(JVAND15):IFJV=0THENJV=K:REM DIRECTION VALUE
4 C=PEEK(V+31)ANDI:D=PEEK(V+30)AND2:IFJV=10THENJV=K:SX=SX-1:IFSX<1THENSX=1
5 IFFR=0THENSX=2
6 IFJV=5THENJV=K:SX=SX+1:IFSX>50THENSX=50
7 IFJV=1THENV=V-SX:POKE2040,200:IFV<0THENV=0
8 IFJV=2THENV=V+SX:POKE2040,200:IFV>255THENV=255
9 IFJV=4THENX=X-SX:POKE2040,210:IFX<0THENX=0
10 IFJV=8THENX=X+SX:POKE2040,209:IFX>320THENX=320
11 IFJV=0THENPOKE2040,206
12 K=JV:IFD=2THENF=2:D=0:GOSUB9000
14 O=PEEK(V+30)ANDB:IFO=8ANDF=2THEND=0:F=0:O=0:GOSUB9000:GOSUB40000:RETURN
19 IFC<>ITHENRETURN
20 IF(TI-RO)<60THENC=0:RETURN
21 GOSUB10000:FORW=YTO225STEP10:POKE2040,Z:POKES+4,17:POKES,127
22 POKES+1,8:POKEV+28,3:FORZ=211TO215:POKE2040,Z:POKES+4,129:POKES+4,6
23 POKEV+1,W:POKES+24,15:FORTI=12TO0STEP-6:POKES+24,TI:NEXT:NEXT
24 NEXT:X=50:Y=225:D=0:POKEV+16,0:POKEV+1,V:POKEV,X:POKEV+28,2:POKE53275,0
25 RETURN
27 POKE53281,15:PRINT"■":CL$="■■■■■■■■":PRINT"■■■■■";TAB(13)"■CAVERN RESCUE
28 GOSUB9000:POKE53280,12:GOSUB50000:GOSUB9000
29 REM START SPRITES AT 13312(BLOCK 208)
30 POKE53281,15:PRINTCHR$(142):POKE52,48:POKE56,48:POKE56334,PEEK(56334)AND254
31 POKE53281,15:PRINTCHR$(142):POKE52,48:POKE56,48:POKE56334,PEEK(56334)AND254
32 POKE1,PEEK(1)AND251:FORI=0TO511:POKEI+12288,PEEK(53248+I):NEXT
33 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
34 POKE53272,(PEEK(53272)AND240)+12
35 FORI=12800TO13015:READA:POKEI,A:NEXT
36 PRINT"■":POKEV+16,1:X=60:Y=200
37 FORI=13312TO13887:READA:POKEI,A:NEXT
38 GOSUB20000:RO=0:GOSUB40030:GOSUB45000
80 RO=TI:POKE2043,216:POKEV+42,8:POKEV+6,40:POKEV+7,225:POKEV+21,PEEK(V+21)OR6
81 POKEV+16,PEEK(V+16)OR8:XK=0:C=0:POKEV+39,8:X=50:Y=225
82 POKEV+21,9:POKE2041,214
83 POKEV+39,8:X=50:Y=225:GOSUB2000
84 POKE2041,INT(RND(0)*4)+211:GOSUB1:IFX>255THENPOKEV+16,9:POKEV,X-255:GOTO86
85 POKEV+16,8:POKEV,X
86 POKEV+1,V:IFC=1THENC=0:GOTO90
87 IFK=5THENK=0:GOTO80
88 IFD=2ANDXK=5THEN90
89 GOTO84
90 PRINT"■":POKEV+16,8:X=50:Y=225:POKEV+1,Y:POKEV,X:POKE2040,200:D=0:XX=0
100 REM DATA FOR  "UDG'S
115 F=0:O=0:D=0:PRINT"■■PRESS FIRE BUTTON FOR A TRY"
116 K=PEEK(56320)AND16:IFK=0THEN80
117 GOTO116
119 REM MUSIC DATA
120 DATA22,96,250,22,96,250,29,233,250,33,135,250,37,162,250,29,223,250
121 DATA33,135,60,33,135,60,33,135,60,22,96,60,22,96,50,29,233,60,33,135
122 DATA60,37,162,60,29,223,60,33,135,60,33,135,60,33,135,200
125 DATA-1,-1,-1
130 REM STALIOTITES
135 DATA255,255,255,253,253,163,163,131
140 DATA255,253,189,189,157,155,139,136
150 DATA255,255,239,231,231,199,135,3
162 DATA92,142,188,142,124,76,76,4
163 DATA255,127,127,63,63,31,15,15
164 DATA255,254,254,252,252,248,240,240
165 DATA15,7,7,3,3,1,1,1
```

Programmation impérative non structurée

How it works

1-25 main joystick and control module

27-35 reset memory and store sprites and user-defined graphics

36-117 main section: switches on sprites, places them on screen and moves them

118-125 theme tune music DATA

130-410 user-defined graphics DATA

5000-5110 top 10 time table — a high-score table for the 10 best times

7000-8750 sprite DATA

9000-9015 bell sound effect

10000-10020 explosion effect

20000-25000 draw cavern routine

30000-30075 winner sprite DATA

40000-40100 win and display routine

45000-45150 instructions

50000-50070 routine to play theme music

60000 random high-resolution colour routine

```
0 V=53248:S=54272:POKE54296,15:FORI=STOS+24:POKEI,0:NEXT:GOTO27
1 V=53248:POKEV+28,2:POKE53275,2:JV=PEEK(56320):IFSX=0THENSX=5
2 POKEV+28,2:POKEV+37,8:POKEV+38,4:FR=JVANDI6:REM FIRE BUTTON STATUS
3 JV=15-(JVAND15):IFJV=0THENJV=K:REM DIRECTION VALUE
4 C=PEEK(V+31)ANDI:D=PEEK(V+30)AND2:IFJV=10THENJV=K:SX=SX-1:IFSX<1THENSX=1
5 IFFR=0THENSX=2
6 IFJV=5THENJV=K:SX=SX+1:IFSX>50THENSX=50
7 IFJV=1THENV=V-SX:POKE2040,200:IFY<0THENY=0
8 IFJV=2THENY=Y+SX:POKE2040,200:IFY>255THENY=255
9 IFJV=4THENX=X-SX:POKE2040,210:IFX<0THENX=0
10 IFJV=8THENX=X+SX:POKE2040,209:IFX>320THENX=320
11 IFJV=0THENPOKE2040,206
12 K=JV:IFD=2THENF=2:D=0:GOSUB9000
14 G=PEEK(V+30)ANDB:IFG=8ANDF=2THEND=0:F=0:G=0:GOSUB9000:GOSUB40000:RETURN
19 IFC<>ITHENRETURN
20 IF(TI-RO)<60THENC=0:RETURN
21 GOSUB10000:FORW=YTO225STEP10:POKE2040,Z:POKES+4,17:POKES,127
22 POKES+1,8:POKEV+28,3:FORZ=211TO215:POKE2040,Z:POKES+4,129:POKES+4,6
23 POKEV+1,W:POKES+24,15:FORTI=12TO0STEP-6:POKES+24,TI:NEXT:NEXT
24 NEXT:X=50:Y=225:D=0:POKEV+16,0:POKEV+1,Y:POKEV,X:POKEV+28,2:POKE53275,0
25 RETURN
27 POKE53281,15:PRINT"■":CL$="■■■■■■■■":PRINT"■■■■■";TAB(13)"■CAVERN RESCUE
28 GOSUB9000:POKE53280,12:GOSUB50000:GOSUB9000
29 REM START SPRITES AT 13312(BLOCK 208)
30 POKE53281,15:PRINTCHR$(142):POKE52,48:POKE56,48:POKE56334,PEEK(56334)AND254
31 POKE53281,15:PRINTCHR$(142):POKE52,48:POKE56,48:POKE56334,PEEK(56334)AND254
32 POKE1,PEEK(1)AND251:FORI=0TO511:POKEI+12288,PEEK(53248+I):NEXT
33 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
34 POKE53272,(PEEK(53272)AND240)+12
35 FORI=12800TO13015:READA:POKEI,A:NEXT
36 PRINT"■":POKEV+16,1:X=60:Y=200
37 FORI=13312TO13887:READA:POKEI,A:NEXT
38 GOSUB20000:RO=0:GOSUB40030:GOSUB45000
80 RO=TI:POKE2043,216:POKEV+42,0:POKEV+6,40:POKEV+7,225:POKEV+21,PEEK(V+21)OR8
81 POKEV+16,PEEK(V+16)OR8:XX=0:C=0:POKEV+39,0:X=50:Y=225
82 POKEV+21,9:POKE2041,214
83 POKEV+39,0:X=50:Y=225:GOSUB2000
84 POKE2041,INT(RND(0)*4)+211:GOSUB1:IFX>255THENPOKEV+16,9:POKEV,X-255:GOTO86
85 POKEV+16,0:POKEV,X
86 POKEV+1,Y:IFC=1THENC=0:GOTO90
87 IFK=5THENK=0:GOTO80
88 IFD=2ANDXX=5THEN90
89 GOTO84
90 PRINT"■":POKEV+16,0:X=50:Y=225:POKEV+1,Y:POKEV,X:POKE2040,200:D=0:XX=0
100 REM DATA FOR  'UDG'S
115 F=0:G=0:D=0:PRINT"■■PRESS FIRE BUTTON FOR A TRY"
116 K=PEEK(56320)AND16:IFK=0THEN80
117 GOTO116
119 REM MUSIC DATA
120 DATA22,96,250,22,96,250,29,233,250,33,135,250,37,162,250,29,223,250
121 DATA33,135,60,33,135,60,33,135,60,22,96,60,22,96,50,29,233,60,33,135
122 DATA60,37,162,60,29,223,60,33,135,60,33,135,60,33,135,200
125 DATA-1,-1,-1
130 REM STALIOTITES
135 DATA255,255,255,253,253,163,163,131
140 DATA255,253,189,189,157,155,139,136
150 DATA255,255,239,231,231,199,135,3
162 DATA192,142,188,142,124,76,76,4
163 DATA255,127,127,63,63,31,15,15
164 DATA255,254,254,252,252,248,240,240
165 DATA15,7,7,3,3,1,1,1
```

Programmation impérative structurée

Deux grandes idées à retenir

- Au niveau global : le raisonnement modulaire
- Dans la conception des boucles : agir directement, au fil de l'eau

Plan

Programmer de façon modulaire

- Pas de longues fonctions, découper en blocs élémentaires.
- Pour chaque fonction un objectif clair
  qui doit pouvoir s'énoncer (se "documenter").
- Réutiliser ses fonctions au maximum.
- Eviter les variables globales : passer l'information en paramètres

Programmer de façon modulaire

- Pas de longues fonctions, découper en blocs élémentaires.
- Pour chaque fonction un objectif clair
  qui doit pouvoir s'énoncer (se "documenter").
- Réutiliser ses fonctions au maximum.
- Eviter les variables globales : passer l'information en paramètres

Eviter les variables globales et leurs effets de bord

Les variables globales produisent des "effets de bord", c'est-à-dire que la réponse de la fonction varie en fonction du "contexte" lors de l'appel. On veut au contraire des blocs au comportement prévisible.

Plan

Préambule X - Complexité (temporelle)

Complexité.
La complexité, ou le temps d'exécution, d'un programme P (fonction ou procédure) est le nombre d'opérations élémentaires (addition, multiplication, affectation, test, etc.) nécessaires à l'exécution de P.
Lorsque cette complexité dépend de deux paramètres n et m, on dira que P a une complexité en O(φ(n,m)) lorsqu'il existe trois constantes A, n0 et m0 telles que la complexité de P est inférieure ou égale à A φ(n,m), pour tout n $\geq$ n0 et m $\geq$ m0.

Lorsqu'il est demandé de donner la complexité d'un programme, le candidat devra justifier cette dernière si elle ne se déduit pas directement de la lecture du code.

Préambule X - Fonctions autorisées

Python.
Dans ce sujet, nous adopterons la syntaxe du langage Python. On rappelle qu'en Python, les listes sont des tableaux dynamiques à une dimension. Sur les listes, on dispose des opérations suivantes, qui ont toutes une complexité constante :
- [] crée une liste vide (c'est-à-dire ne contenant aucun élément).
- len(liste) renvoie la longueur de la liste liste.
- liste.append(x) ajoute l'élément x à la fin de la liste liste.
- liste[i] renvoie le (i + 1)-ième élément de la liste liste s'il existe ou produit une erreur sinon (noter que le premier élément de la liste est liste[0]).

L'expression [k for i in range(n)] construit une liste de longueur n contenant n occurrences de k.
Important : l'utilisation de toute autre fonction sur les listes telle que liste.insert(i,x), liste.remove(x), liste.index(x), ou encore liste.sort(x) est rigoureusement interdite. Ces fonctions devront être réécrites explicitement si nécessaire.

Plan

Pas de test superflu

Aux concours, par défaut : la donnée est "conforme" à la description.

- **Faire des contrôles seulement si demandé.**

  Outil possible : `assert` interrompt l'exécution suite à un test

  syntaxe `assert MachinEstCommeIlFaut`

Bien sûr c'est spécifique à cette année ! En vrai... on doit tout surveiller !

Pas de modification des données, sauf demande

▷ **Ne jamais modifier ce qu'on vous donne, sauf si demandé**

La règle : vous pouvez "lire uniquement"
- ne pas vider une liste même "si ça ne sert plus"
- ne pas rallonger une liste même avec des "0"

L'exception : si on vous confie une liste pour modification

Exemple : échanger deux éléments, trier la liste...

Optimiser, avec discernement

▷ **On veut surtout gagner des ordres de grandeur**

- compliquer beaucoup le code pour un facteur 2 ou 3 : bof
- un gain substantiel $O(n^2) \to O(n \ln n)$: oui

Dans la "vraie" pratique, il faut aussi regarder les constantes cachées dans le O(.), faire des tests concrets.

Pas de mémorisation inutile

▷ **Repérer et supprimer la mémorisation inutile**

- économie d'espace : comme pour le temps
- utiliser 2,3 ou 5 variables entières ne change pas grand chose.
- stocker sans utilité n variables dans une liste est répréhensible.

Pas de virtuosité syntaxique

▷ **privilégier des écritures "standard"**

- pour la lisibilité, la portabilité,
- aussi parfois l'efficacité ("raccourcis" à la complexité pas toujours claire).

Discerner
- raccourcis Python raisonnables `a,b=b,a`, `a+=1`.
- spécificités trop Pythonnesques : conversion automatique, gestion des ambigüités des noms ... Ne pas employer

Plan

Signature et spécification

Rappel : signature et spécification

Un énoncé vous propose de coder en respectant
- une signature (type de données d'entrées / type de données de sorties)
- une spécification : description de l'effet (notamment les valeurs de sortie).

Quand ce travail est bien fait il décrit très précisément l'effet, sans laisser d'ambigüité.

Comment documenter

En théorie on devrait toujours spécifier les fonctions !

Rappel : comment faire : le "docstring" :

chaîne de caractère entre """ et """ sur la ligne qui suit le def
qui apparaît quand on utilise help.

Ne pas confondre avec les commentaires (♯)
- au fil du programme,
- utiles pour recomprendre le code : grandes parties, choix atypiques...
- mais sans faire de paraphrase

Les "vrais" programmeurs documenteront leur code de façon très systématique et beaucoup plus structurée.