

INF1035

Expressions et variables numériques

Semaine 3

- Expressions
- Variables
- Affectation
- Calculs numériques

Commentaires

Les commentaires

- Les langages de programmation fournissent une méthode pour l'insertion de commentaires au sein du code afin de fournir des informations supplémentaires:
 - **Métadonnées sur le code:** nom développeur, dernière date de modification, version...
 - **Commenter/expliciter** une partie d'un programme pour faciliter sa compréhension aux autres développeurs ou pour une modification ultérieure plus aisée,
- Un commentaire n'est autre qu'un texte qui sera ignoré lors de l'exécution du programme ☾ ils ont le même effet que des espaces blancs

Deux types de commentaires

- **En une seule ligne:** précédé par **dièse #**

```
1  
2  #ceci est un commentaire  
3  print("bonjour") #tout ce qui se trouve après le diez sera ignoré
```

- **Sur plusieurs lignes:** commence et se termine par **''' (3 apostrophes)**

```
'''  
Bonjour tout le monde  
Version:1  
Date: 14/01/2022  
'''
```

Syntaxe et expression

Syntaxe

- **Syntaxe d'un langage** : forme textuelle que peuvent prendre les programmes valides
- Tout comme pour les langages naturels (français, anglais, ...), la syntaxe est normalement définie par une **grammaire**
- **Grammaire** : ensemble de règles pour former des programmes valides syntaxiquement à partir de fragments de programme valides

Expressions (1)

- Tous les langages de programmation offrent la possibilité de faire des calculs numériques
- Ces calculs s'expriment par des expressions

Exemple:

- 5
- $2 + 3 * 5$
- $(2 + 3) * 5$

Expressions (2)

- Toute expression a une valeur, qui est le résultat du calcul exprimé par l'expression
- En Python, **25** est la valeur de l'expression **(2 + 3) * 5**
- Dans presque tous les langages, un nombre décimal non-négatif est une **expression simple** (une constante littérale), dont la valeur est le nombre en question
- En Python, **123** est la valeur de l'expression **123**

Expressions (3)

- Des expressions plus complexes sont bâties à l'aide d'opérateurs et d'expressions plus simples (les opérandes)
- Les opérateurs de base en **Python**:
 - **+** **addition**
 - **-** **soustraction**
 - ***** **multiplication**
 - **/** **division**
 - ****** **exponentiation (puissance)**
 - **%** **reste de divisions euclidiennes**
 - **//** **division entière**

Opérateurs binaires

- "Binaire" pour 2 opérandes

Syntaxe: *<expression>* *<op>* *<expression>*

Exemples:

3 + 7	☾	10
2 * 5 - 1	☾	9
2 * 5 - 3 * 3	☾	1

Opérateurs unaires

- Les opérateurs de signe (+ et -) peuvent être utilisés comme préfixe d'une expression

Syntaxe: *<signe> <expression>* (+ et -) unaires

Exemples

-5

☾ - 5

+13

☾ 13

--5

☾ 5

Préséance des opérateurs (1)

- Chaque opérateur a un niveau de préséance (ou précédence)
 - **+, -** : les opérateurs Binaires additifs (**niveau 1**)
 - ***, /, //, %** : les opérateurs Binaires multiplicatifs (**niveau 2**)
Pour déterminer comment les sous-expressions se regroupent, il faut regrouper les sous-expressions aux côtés des opérateurs de niveau 2 avant de le faire pour les opérateurs de niveau 1.
 - Les opérateurs unaires **+** et **-** sont de **niveau 3**
 - ******: l'opérateur d'exponentiation est de **niveau 4**
- Pour forcer un groupement spécifique, on peut se servir de parenthèses (les parenthèses sont de **niveau 5**) (le plus prioritaire)

Préséance des opérateurs (2)

Exemples:

○ $1 + 2 * 3$

☾ $1 + (2 * 3) = 7$

○ $-+-5$

☾ $-(+(-5)) = 5$

○ $- 8 * -5 -3$

☾ $((-8) * (-5)) - 3 = 37$

○ $-(-2) ** 2$

☾ $-((-2) ** 2) = -4$

○ $11 \% 3 - 1$

☾ $(11 \% 3) - 1 = 1$

Associativité des opérateurs (1)

- Pour des opérateurs de même niveau de préséance, pour déterminer comment les sous-expressions se regroupent, il faut tenir compte de l'associativité des opérateurs
- Les opérateurs $+, -, *, /$ sont associatifs à gauche **1 - 2 + 3** est égal à **(1 - 2) + 3** mais pas à **~~1 - (-2 + 3)~~**

Exemple:

$$1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 = (((((((1 - 2) - 3) - 4) - 5) - 6) - 7) - 8) - 9$$

Associativité des opérateurs (2)

- Certaines paires de parenthèses dans une expression peuvent être **redondantes** (c'est-à-dire qu'on obtient le même regroupement lorsqu'on les retirent)

Exemple:

$$(8 * 9) / (7 - 1) = 8 * 9 / (7 - 1)$$

- Si ça aide à **comprendre la logique** du calcul, il est bon de **garder** des parenthèses redondantes.

Exemple:

$$(- 8) + (- 5) \text{ au lieu de } - 8 + - 5$$


Erreurs de syntaxe

- L'interprète fait l'**analyse syntaxique** du code avant de l'exécuter
- Si le code ne correspond pas à la grammaire de Python un **message d'erreur** sera affiché

```
➤ 1 + 2 x 3
  File "<stdin>", line 1
    1 + 2 x 3
          ^
SyntaxError: invalid syntax

➤ 1 + 2) * 3
  File "<stdin>", line 1
    1 + 2) * 3
          ^
SyntaxError: unmatched ') '

➤ ((1 + 2) + 3) + 4
... 
```



Expression incomplète

Les nombres

Les nombres en Python (1)

- La syntaxe des nombres permet de préciser des **décimales** et une **puissance de 10**

Exemples

○ $1.25 = 1.25$

○ $42e3 = 42 \times 10^3 = 42000.0$

○ $.2e-1 = 0.2 \times 10^{-1} = 0.2 \times 10^1 = 0.02$

○ $1.030E+10 = 1.03 \times 10^{10} = 10300000000.0$

Les nombres en Python (2)

- L'affichage d'un nombre qui n'a pas de partie fractionnaire ne contient **pas** de zéros à la fin.

```
> 15
15
> 15 * 2
30
> 15 * 2.1
31.5
>
```

- Pour les nombres $\geq 10^{15}$, l'affichage se fait avec la **notation scientifique**.

```
> 10E14
10000000000000000.0
> 10E15
1e+16
>
```

Abstraction

Un texte

- Examinons le texte suivant:

«Le fils de Rose-Anne Monna et de Legrand Feeley qui réside à Montréal a acheté un télescope au troisième mois de 2012. Le fils de Rose-Anne Monna et de Legrand Feeley qui réside à Montréal a observé la quatrième planète en orbite autour du Soleil.»

- Ce texte est plutôt lourd...

- Que peut-on faire pour l'alléger?

Abstraire en nommant (1)

- Utilisons les noms propres pour abstraire:
«**Marc** a acheté un télescope en **Mars** 2012. **Marc** a observé **Mars**.»
- Le texte est beaucoup plus court, agréable à lire et compréhensible
- Les noms prennent le sens de leur définition (**p.ex.** Marc = «Le fils de *Rose-Anne Monna* et de *Legrand Feeley* qui réside à Montréal»)

Abstraire en nommant (2)

- En programmation, les noms sont des **identificateurs**, et on en donne la définition dans une **déclaration**
- Lorsqu'on **réfère** à un identificateur, c'est une **déclaration spécifique** à laquelle on fait référence.
- Les ambiguïtés possibles, comme pour **Mars**, sont réglées par le **contexte de la référence** c'est-à-dire où et comment la référence est faite (**p.ex.** grâce **aux règles de portée**)

Syntaxe des identificateurs (1)

- En Python, les identificateurs sont des symboles composés de lettres (majuscules/minuscules), des chiffres (0-9), et le caractère _

Exemple:

julie, mars1, M0nna, legrand_Feeley ☾ **valides**
garçon, Elève, bonjour!, temps-max ☾ **invalides**

- Les chiffres sont **interdits** au début d'un identificateur

Exemple:

0Monna ☾ **invalide**
M0nna, monna1 ☾ **valides**

Syntaxe des identificateurs (2)

- La casse (majuscule/minuscule) est significative

Exemple:

Marc, marc, marC et *MARC* sont traités comme **identificateurs différents**

- à l'exception du **caractère** `_` Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que `$`, `#`, `@`, etc. sont interdits.

Exemple:

garçon, Elève, bonjour!, temps-max ☾ **invalides**

Syntaxe des identificateurs (3)

- En plus de ces règles, il faut encore ajouter que vous ne pouvez pas utiliser comme noms de variables les **29 «mots réservés»** au langage ci-

and	assert	break	class	continue	def
del	elif	else	except	exec	finally
for	from	global	if	import	in
is	lambda	not	or	pass	print
raise	return	try	while	yield	

Choix de nom d'un identificateur

- Un **bon** identificateur clarifie **ce à quoi il réfère** (il évite les ambiguïtés)
- Si la déclaration peut être référée de partout dans un gros programme, il est mieux d'utiliser un identificateur le plus descriptif possible:

Exemple:

temperature_congelation_hydrogen, temp_cong_hydrogene

- Si la portée est locale, il est mieux d'utiliser un identificateur court pour alléger le code :

Exemples:

x, var, longueur

Affectation

Affectation (1)

- La programmation impérative est intimement liée au concept d'état de la machine, et de **modification d'état**
- **L'affectation (assignement)** est une opération qui change la valeur contenue dans une cellule mémoire, comme celle associée à une variable

Syntaxe: *⟨identificateur⟩ = ⟨expression⟩*
⟨identificateur⟩ est le nom de la variable créée

- La valeur de *⟨expression⟩* vient remplacer la valeur dans la cellule associée à *⟨identificateur⟩*

Affectation (2)

- Exemple:

```
>>> n = 9  
>>> x = 1
```



Affectation (2)

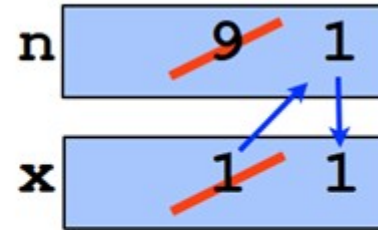
- Exemple:

```
>>> n = 9  
>>> x = 1  
>>> n = x+12
```



Affectation (3)

- Exemple:



```
print("x = "+str(x))=> x = 1
```

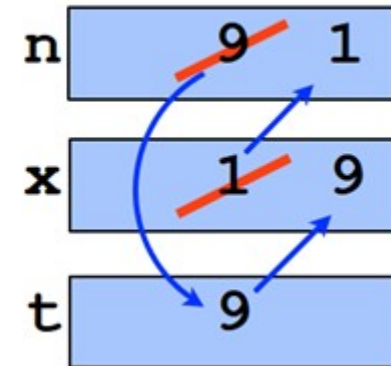
```
print("n = "+str(n))=> n = 1
```


Affectation (4)

- Exemple:

```
n = 9
x = 1

t = n
n = x
x = t
```



```
print("x = ", x )=> x = 9
print("n = "+str(n))=> n = 1
print("t = "+str(t))=> t = 9
```

Affectations multiples

- Pour affecter la même valeur à deux variable x et y

```
x = y = 7
x
7
y
7
□
```

- On peut aussi effectuer des **affectations parallèles** à l'aide de

```
a, b = 4, 8.33
a
4
b
8.33
□
```

- Dans cet exemple **a** et **b** prennent simultanément les nouvelles valeurs **4** et **8.33**

Affectation de variable – Exemple (1)

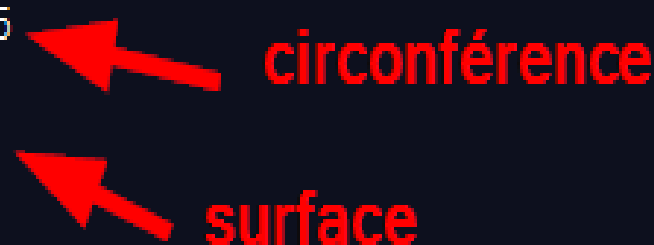
- Problème: calcul de la circonférence et de la surface d'un cercle de rayon 5

Rappel:

circonférence = $2 * \text{Pi} * r$, surface = $\text{Pi} * r * r$

Solution: sans variables

```
> 2 * 3.141592653589793 * 5
31.41592653589793
> 3.14159265398783 * 5 * 5
78.53981634969574
> 
```

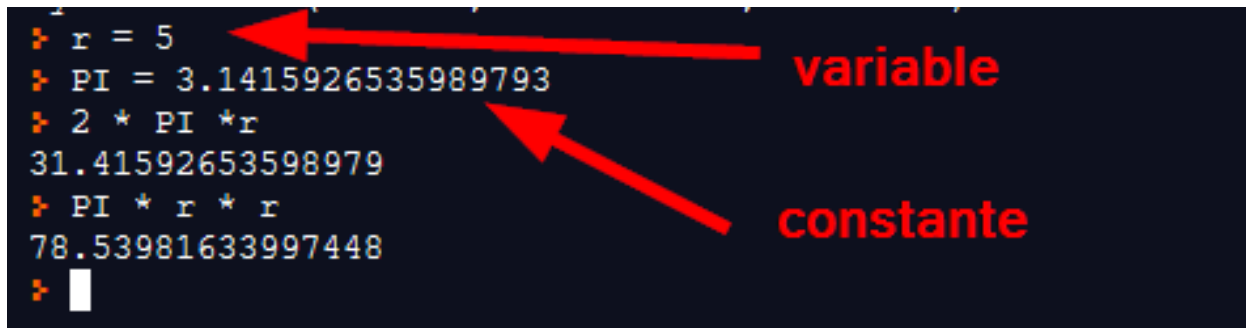


← circonférence

← surface

Affectation de variable – Exemple (2)

- ***Solution:*** en utilisant des variables / constantes



```
➤ r = 5
➤ PI = 3.1415926535989793
➤ 2 * PI * r
31.41592653598979
➤ PI * r * r
78.53981633997448
➤
```

variable (points to `r = 5`)

constante (points to `PI = 3.1415926535989793`)

- En utilisant les variables, la solution devient:
 - Plus lisible
 - Plus facile à comprendre
 - Plus facile à maintenir (change le rayon ou la précision de Pi)
 - Correcte (la première solution a un bogue à cause de la duplication de code)

- **Principe: éviter la duplication de code**

Autres opérateurs d'affectation (1)

- En Python, les opérateurs arithmétiques ont une version travaillant "en-place": +=, -=, *=, /=, **=, %=, //=

☐ `a += b`

☾ `a = a + b`

☐ `a -= b`

☾ `a = a - b`

☐ `a *= b`

☾ `a = a * b`

☐ `a /= b`

☾ `a = a / b`

☐ `a **= b`

☾ `a = a ** b`

☐ `a %= b`

☾ `a = a % b`

☐ `a //= b`

☾ `a = a // b`

Autres opérateurs d'affectation (2)

- En Python, les opérateurs arithmétiques ont une version travaillant "en-place": +=, -=, *=, /=, **=, %=, //=

Exemple: b=2, a=9 ☾ nouvelle valeur de a?

☐ a += b

☾ a = a + b

☐ a -= b

☾ a = a - b

☐ a *= b

☾ a = a * b

☐ a /= b

☾ a = a / b

☐ a **= b

☾ a = a ** b

☐ a %= b

☾ a = a % b

☐ a //= b

☾ a = a // b

Autres opérateurs d'affectation (2)

- En Python, les opérateurs arithmétiques ont une version travaillant "en-place": +=, -=, *=, /=, **=, %=, //=

Exemple: b=2, a=9 € nouvelle valeur de a?

○ **a += b**
11

€ **a = a + b**

○ **a -= b**

€

a = a - b

7

○ **a *= b**
18

€

a = a * b

○ **a /= b**

€

a = a / b

4.5

○ **a **= b**

€

a = a ** b

81

○ **a %= b**

€

a = a % b

1

○ **a //= b**

€

a = a // b

4

Calculs numériques

Fonctions mathématiques (1)

- Python possède des **fonctions prédéfinies** qui correspondent à des fonctions mathématiques bien connues
- Il faut faire **import math** tout d'abord
- Ensuite utiliser **math.nom_fonction()** par exemple

```
> import math
> math.ceil(3.8)
4
> math.floor(4.7)
4
```

Fonctions mathématiques (2)

- **`math.floor(-7.6)`** ☾ partie entière, plancher, donne ici -8.0.
- **`int(math.floor(4.5))`** ☾ pour avoir l'entier 4.
- **`math.ceil(-7.6)`** ☾ entier immédiatement supérieur, plafond, donne ici -7.
- **`math.exp(2)`** ☾ exponentielle,
- **`math.log(2)`** ☾ logarithme en base naturelle
- **`math.log10(2)`** ☾ logarithme en base 10.
- **`math.log(8, 2)`** ☾ log de 8 en base 2.

```
> import math
> math.exp(2)
7.38905609893065
>
```

Fonctions mathématiques (3)

- **`math.sqrt(9)`** € racine carrée.

```
➤ math.sqrt(9)  
3.0
```

- **`math.pow(4, 5)`** € 4 puissance 5.

- **`math.fmod(4.7, 1.5)`** € modulo, ici 0.2.

```
➤ math.fmod(4.7, 1.5)  
0.200000000000000018
```

- Préférer cette fonction à % pour les flottants.

- **`math.factorial(4)`** € factorielle 4, donc 24

(uniquement pour les entiers positifs)

```
➤ math.fsum([2 for i in range(3)])  
6.0
```

- **`math.fsum([2 for i in range(3)])`** :

€ fait la somme de l'élément 2 trois fois

- **`math.fsum([2,3,4])`** € fait la somme des éléments

```
➤ math.fsum([2,3,4])  
9.0
```

Fonctions mathématiques (4)

- **math.isinf(x)** ☾ teste si x est infini (**inf**) et renvoie True si c'est le cas.
- **math.isnan(x)** ☾ teste si x est nan (**Not a Number**) et renvoie **True** si c'est le cas. (**N.B:** True ou Vrai est une valeur Booléenne, à voir dans un prochain cours)
- **fonctions trigonométriques** ☾ **math.sin, math.cos, math.tan, math.asin, math.acos, math.atan** (l'argument est en radians).
- **fonctions hyperboliques** ☾ **math.sinh, math.cosh, math.tanh, math.asinh, math.acosh, math.atanh**
- **math.degrees(x)** ☾ convertit de radians en degrés (**math.radians(x)** pour l'inverse).

Fonctions mathématiques (5)

- **Random** permet la génération de nombres aléatoires.
- Pour pouvoir l'utiliser, on doit commencer par l'importer

import random

Exemples:

import random

rnd = random.Random()

rnd.choice(['a', 'b', 'c'])

rnd.choice([9, 18, 1])

```
> import random
> rnd = random.Random()
> rnd.choice(['a', 'b', 'c'])
'a'
> rnd.choice([9, 18, 1])
9
```

- Valeurs aléatoires :

○ **random.random()** : valeur entre 0 et 1, 1

```
> random.random()
0.8074016508328891
```

○ **random.randint(0, 3)** : entier entre 0 et 3

```
> random.randint(0, 3)
0
```

Exercices

Exercices (1)

● Exercice 1:

Décrivez le plus clairement et le plus complètement possible ce qui se passe à chacune des trois lignes de l'exemple ci-dessous :

```
>>> largeur = 20
```

```
>>> hauteur = 5 * 9.3
```

```
>>> largeur * hauteur
```

```
930
```

Exercices (2)

● Exercice 2:

Assignez les valeurs respectives 3, 5, 7 à trois variables a, b, c.

Effectuez l'opération $a - b/c$.

● Exercice 3:

Mettez dans une variable votre taille en pouce,
Affichez ensuite cette mesure convertie en cm

1 pouce = 2,54cm