Lycée Victor Hugo
MATHÉMATIQUES
BCPST 1

Formulaire Python 3

Opérations mathématiques

Opération	Symbole Python	Exemples
Addition	`+`	`1 + 1` → `2`
Soustraction	`-`	`1 - 1` → `0`
Multiplication	`*`	`2 * 3` → `6`
Puissance	`**`	`2**3` → `8`
Division	`/`	`5 / 2` → `2.5`
Division entière	`//`	`5 // 2` → `2`
Modulo	`%`	`5 % 2` → `1`

Types de données

Type	Nom Python	Exemples
entier	`int`	`234` `0` `-32`
réel	`float`	`12.34` `2.0` `-1.7e-6`
booléen	`bool`	`True` `False`
chaine	`str`	`"abc"` `"Bonjour !\nComment ca va ?"`
liste	`list`	`[1,2,3]` `[0.1, 4, "abc", True]`
dictionnaire	`dict`	`{'nom': "Hugo", 'prénom': "Victor", 'age': 83}`

Affectation de variable

Type d'affection	Syntaxe Python	Exemple
Affection simple	`nom_variable = valeur`	`x = 2 + 3.4`
Affection multiple	`variable_1 = ... = variable_n = valeur_commune`	`x = y = z = 0`
Affection simultanée	`variable_1, ...,variable_n = valeur_1, ..., valeur_n`	`x, y = 2, 3.4`

Raccourcis pour modifier une variable

Syntaxe	Syntaxe raccourcie	Exemple
`variable = variable + valeur`	`variable += valeur`	`x += 2`
`variable = variable - valeur`	`variable -= valeur`	`x -= 1`
`variable = variable * valeur`	`variable *= valeur`	`x *= 2`
`variable = variable ** valeur`	`variable **= valeur`	`x **= 1`
`variable = variable / valeur`	`variable /= valeur`	`x /= 1`
`variable = variable // valeur`	`variable //= valeur`	`x //= 1`
`variable = variable % valeur`	`variable %= valeur`	`x %= 1`

Conversion de type

Le nom du type fait office de fonction de conversion :

```
>>> int(3.14)        >>> float(2)        >>> str(1234)        >>> int("18")
3                    2.0                 "1234"               18
```

Fonctions mathématiques

À l'aide de la commande `from math import sin, cos, pi, ...` ou `from math import *`, on peut utiliser les fonctions et constantes mathématiques suivantes :

Fonction	Équivalent mathématique	Fonction	Équivalent mathématique
`sqrt(x)`	$\sqrt{x}$	`exp(x)`	e^x
`log(x)`	$\ln x$	`log10(x)`	$\log x$
`cos(x)`	$\cos x$	`acos(x)`	$\arccos x$
`sin(x)`	$\sin x$	`asin(x)`	$\arcsin x$
`tan(x)`	$\tan x$	`atan(x)`	$\arctan x$
`floor(x)`	$\lfloor x \rfloor$	`factorial(n)`	$n!$
`comb(n, k)`	$\binom{n}{k}$		

Constante	Équivalent mathématique
`e`	e
`pi`	π
`inf`	$+\infty$

Logique booléenne

Test	Syntaxe Python	Exemples
Inférieur et supérieur strict	`<` et `>`	`x < 2` `x > 2`
Inférieur et supérieur ou égal	`<=` et `>=`	`x <= 2` `x >= 2`
Égalité et différence	`==` et `!=`	`2 == 1+1` `x != 1`
Appartenance	`in`	`"a" in "abc"` `1 in [1,2,3]`
Connecteurs ET et OU	`and` et `or`	`(x<2) and (y>3)` `(x<2) or (y>3)`
Négation	`not`	`not((x == 1) or (x > 2))`

Chaînes de caractères

Fonction	Syntaxe Python	Type de fonction
Longueur	`len(chaine)`	`str` → `int`
Afficher	`print(chaine)`	`str` → `None`
Séparer une chaine	`chaine.split(séparateur)`	`str` → `list`
Requête utilisateur	`reponse = input(question)`	`str` → `str`

Listes

Fonction	Syntaxe Python	Type de fonction
Longueur	`len(liste)`	`list` → `int`
Minimum	`min(liste)`	`list` → `int`
Maximum	`max(liste)`	`list` → `int`
Somme	`sum(liste)`	`list` → `int`/`float`
Position	`liste.index(valeur)`	`object` → `int`
Nombre d'occurrences	`liste.count(valeur)`	`object` → `int`
Ajout d'un élément final	`liste.append(valeur)`	`object` → `None`
Insérer un élément	`liste.insert(indice, valeur)`	`int`×`object` → `None`
Supprimer un élément	`liste.remove(valeur)`	`object` → `None`
Supprimer un élément	`liste.pop(indice)`	`int` → `object`
Trier	`liste.sort()`	`None` → `None`
Renverser la liste	`liste.reverse()`	`None` → `None`

Instruction conditionnelle, tests

Type de test	Syntaxe Python
Test simple	`if condition:` `    bloc d'instructions`

Test avec alternative	`if condition:`
	`    bloc d'instructions`
	`else:`
	`    bloc d'instructions`

Test avec plusieurs alternatives	`if condition_1:`
	`    bloc d'instructions`
	`elif condition_2:`
	`    bloc d'instructions`
	`    ⋮`
	`elif condition_n:`
	`    bloc d'instructions`
	`else:`
	`    bloc d'instructions`

Boucle for

```
for i in range(n):
    bloc d'instructions      # i varie de 0 à n-1 (le bloc est exécuté n fois)

for i in range(a, b):
    bloc d'instructions      # i varie de a à b-1 (le bloc est exécuté b-a fois)

for e in liste:
    bloc d'instructions      # e prend les valeurs des éléments successifs de liste
```

Boucle while

```
while condition:
    bloc d'instructions
```

Fonctions

```
def nom_fonction(nom_paramètre_1, nom_paramètre_2, ..., nom_paramètre_n):
    bloc d'instructions
```