Cheatsheet C# vs Python

Variables et Types de Données

Concept	C#	Python
Déclaration	Type explicite: `int age = 30;`	Type dynamique: `age = 30`
Entier	`int nombre = 42;`	`nombre = 42`
Flottant	`float x = 2.5f;` `double y = 3.14;`	`x = 2.5`
Chaîne	`string nom = "Alice";`	`nom = "Alice"`
Booléen	`bool estVrai = true;`	`est_vrai = True`
Constante	`const int MAX = 100;`	`MAX = 100` (convention)
Liste	`List<int> nombres = new List<int>{1, 2, 3};`	`nombres = [1, 2, 3]`
Dictionnaire	`Dictionary<string, int> ages = new Dictionary<string, int>();`	`ages = {}` ou `ages = dict()`
Type nullable	`int? x = null;`	N/A (tous les types sont "nullables")

Opérateurs

Opération	C#	Python
Addition	`a + b`	`a + b`
Soustraction	`a - b`	`a - b`
Multiplication	`a * b`	`a * b`
Division	`a / b`	`a / b` (toujours flottant)
Division entière	`a / b` (si a et b sont int)	`a // b`
Modulo	`a % b`	`a % b`
Puissance	`Math.Pow(a, b)`	`a ** b`
Incrémentation	`a++` ou `++a`	`a += 1`
Décrémentation	`a--` ou `--a`	`a -= 1`
Égalité	`a == b`	`a == b`
Inégalité	`a != b`	`a != b`
Supérieur	`a > b`	`a > b`

Opération	C#	Python
Inférieur	`a < b`	`a < b`
Supérieur ou égal	`a >= b`	`a >= b`
Inférieur ou égal	`a <= b`	`a <= b`
ET logique	`a && b`	`a and b`
OU logique	`a || b`	`a or b`
NON logique	`!a`	`not a`

Structures de Contrôle

Conditions

C#:

```
if (condition) {
    // code
} else if (autreCondition) {
    // code
} else {
    // code
}
```

Python:

```
if condition:
    # code
elif autre_condition:
    # code
else:
    # code
```

Switch/Match

C#:

```
switch (valeur) {
    case 1:
        // code
        break;
    case 2:
        // code
        break;
    default:
```

```
        // code
        break;
}
```

Python (3.10+):

```
match valeur:
    case 1:
        # code
    case 2:
        # code
    case _:
        # code par défaut
```

Boucle While

C#:

```
while (condition) {
    // code
}
```

Python:

```
while condition:
    # code
```

Boucle For

C#:

```
for (int i = 0; i < 10; i++) {
    // code
}
```

Python:

```
for i in range(10):
    # code
```

Foreach

C#:

```
foreach (var item in collection) {
    // code
}
```

Python:

```
for item in collection:
    # code
```

Fonctions

C#:

```
public int Addition(int a, int b) {
    return a + b;
}
```

Python:

```
def addition(a, b):
    return a + b
```

C# avec paramètres optionnels:

```
public int Addition(int a, int b = 0) {
    return a + b;
}
```

Python avec paramètres par défaut:

```
def addition(a, b=0):
    return a + b
```

Entrées/Sorties Console

C#:

```
// Affichage
Console.WriteLine("Bonjour");

// Lecture
string entree = Console.ReadLine();
int nombre = Convert.ToInt32(Console.ReadLine());
```

Python:

```
# Affichage
print("Bonjour")

# Lecture
entree = input()
nombre = int(input())
```

Conversion de Types

C#:

```
int i = 42;
string s = i.ToString();       // Entier vers chaîne
int j = Convert.ToInt32(s);    // Chaîne vers entier
double d = Convert.ToDouble(s); // Chaîne vers double
```

Python:

```
i = 42
s = str(i)      # Entier vers chaîne
j = int(s)      # Chaîne vers entier
d = float(s)    # Chaîne vers float
```

Lecture/Écriture de Fichiers

C#:

```
// Lecture
string contenu = File.ReadAllText("fichier.txt");
string[] lignes = File.ReadAllLines("fichier.txt");

// Écriture
File.WriteAllText("fichier.txt", contenu);
File.WriteAllLines("fichier.txt", lignes);
```

Python:

```python
# Lecture
with open("fichier.txt", "r") as fichier:
    contenu = fichier.read()

with open("fichier.txt", "r") as fichier:
    lignes = fichier.readlines()

# Écriture
with open("fichier.txt", "w") as fichier:
    fichier.write(contenu)

with open("fichier.txt", "w") as fichier:
    fichier.writelines(lignes)
```

Gestion des Erreurs

C#:

```csharp
try {
    // code pouvant générer une exception
} catch (Exception ex) {
    // gestion de l'erreur
    Console.WriteLine(ex.Message);
} finally {
    // code exécuté dans tous les cas
}
```

Python:

```python
try:
    # code pouvant générer une exception
except Exception as ex:
    # gestion de l'erreur
    print(str(ex))
finally:
    # code exécuté dans tous les cas
```