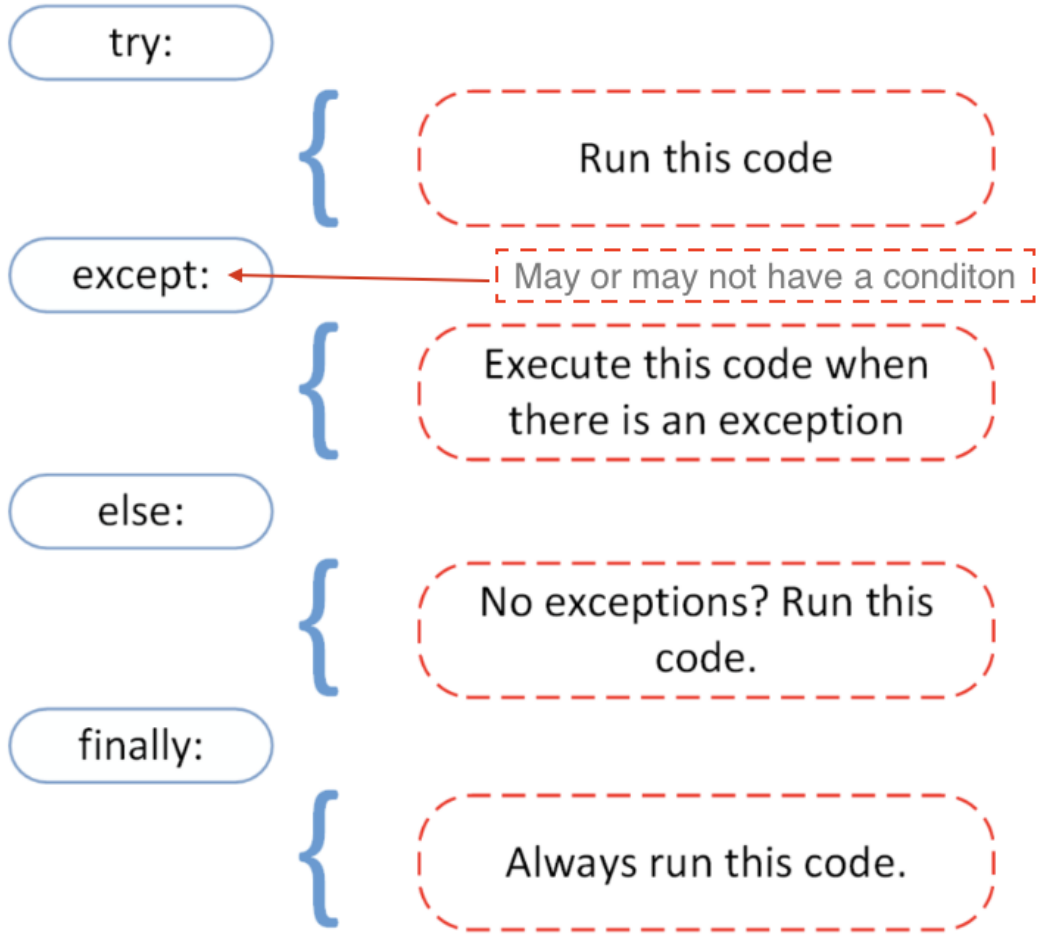
Jour 17 Gestion des exceptions

Gestion des exceptions

Python utilise *try* et *except* pour gérer gracieusement les erreurs. Une sortie gracieuse (ou une manipulation gracieuse) des erreurs est un simple idiome de programmation - un programme d étecte une condition d'erreur grave et "sort gracieusement", de manière contrôlée en conséquen ce. Souvent, le programme imprime un message d'erreur descriptif à un terminal ou un journal dans le cadre de la sortie gracieuse, ce qui rend notre application plus robuste. La cause d'une e xception est souvent externe au programme lui-même. Un exemple d'exceptions pourrait être u ne entrée incorrecte, un mauvais nom de fichier, incapable de trouver un fichier, un périphériqu e IO défectueux. La manipulation gracieuse des erreurs empêche nos applications de s'écraser.

Nous avons couvert les différents types Python *error* dans la section précédente. Si nou s utilisons *try* et *except* dans notre programme, il n'augmentera pas les erreurs dans ces blocs.



Essayez: code dans ce bloc si les choses se passent bien sauf: code da ns ce bloc Exécutez si les choses tournent mal

Exemple:

```
Essayez: imprimer (10 + '5') sauf: print ('quelque c
hose s'est mal passé')
```

Dans l'exemple ci-dessus, le deuxième opérande est une chaîne. Nous pourrions le changer e n flottant ou int pour l'ajouter avec le numéro pour le faire fonctionner. Mais sans aucun chan gement, le deuxième bloc, *except*, sera exécuté.

Exemple:

```
Essayez: Nom = entrée ('Entrez votre nom:') Year_born = entrée ('Année Vous
êtes né:') Age = 2019 - Year_born Print (F'you Are {Name}. Et votre âge est {A
ge}. ') Sauf: Print (' quelque chose qui a mal tourné ').
```

```
Quelque chose s'est mal passé
```

Dans l'exemple ci-dessus, le bloc d'exception s'exécutera et nous ne connaissons pas exacte ment le problème. Pour analyser le problème, nous pouvons utiliser les différents types d'err eur avec sauf.

Dans l'exemple suivant, il gérera l'erreur et nous dira également le parent de l'erreur soulevée .

```
Essayez: Nom = entrée ('Entrez votre nom:') Year_born = entrée ('Année Vous
êtes né:') Age = 2019 - Year_born Print (F'you Are {Name}. Et votre âge est {A
ge}. ') SauftError: Print (' Type Error Occured ') Saufror: Imprime erreur s'est pr
oduite ') Entrez votre nom: Asabeneh Année que vous êtes né: 1920 Type Erreu
r s'est produite
```

Dans le code ci-dessus, la sortie sera *TypeError*. Maintenant, ajoutons un bloc supplémenta ire:

```
essayer:
Nom = entrée ('Entrez votre nom:') année_born = entrée ('année vous né:') Âge
= 2019 - int (année_born) imprimer (f'you est {nom}. Et votre âge est {âge}. ')
SauftError: print (' Type Error se produit ') SaufrorError: imprimer (' une erreur
de valeur se produit ') Saufror se produit ') else: imprimez (' je cours habituellem
ent avec le bloc d'essai ') enfin: imprimer (' je cours toujours. ')
```

```
Entrez votre nom: Asabeneh Year vous né: 1920 Vous
êtes Asabeneh. Et votre âge a 99 ans.

Je cours habituellement avec le bloc d'essai que je
cours toujours.
```

Il raccourcit également le code ci-dessus comme suit:

```
Essayez: Nom = entrée ('Entrez votre nom:') Year_born = entrée ('année vous n
é:') Age = 2019 - int (année_born) print (f'you est {nom}. Et votre âge est {âge}
. ') sauf exception comme e: print (e)
```

Emballage et déballage des arguments en python

Nous utilisons deux opérateurs:

- * Pour les tuples
- ** pour les dictionnaires

Prenons comme exemple ci-dessous. Il ne faut que des arguments mais nous avons la liste. Nous pouvons déballer la liste et les modifications de l'argument.

Déballage

Listes de déballage

```
def sum_of_five_nums (a, b, c, d, e):
```

```
        renvoie un + b + c + d + e

lst = [1, 2, 3, 4, 5]
print (sum_of_five_nums (lst)) # typeError: sum_of_five_nums () manquant 4 arguments de position requis: 'b', 'c', 'd' et 'e'
```

Lorsque nous exécutons ce code ce code, il soulève une erreur, car cette fonction prend des nombres (pas une liste) comme arguments. Décomposons / détruisons la liste.

```
def sum_of_five_nums (a, b, c, d, e): return a + b + c +
d + e

lst = [1, 2, 3, 4, 5] imprimer (sum_of_five_nums (* lst
)) # 15
```

Nous pouvons également utiliser le déballage dans la fonction intégrée de la plage qui s'atten d à un début et à une fin.

```
Nombres = plage (2, 7) # appel normal avec des arguments séparés imprimer (liste (nombre s)) # [2, 3, 4, 5, 6] args = [2, 7]

Nombres = gamme (* args) # Appel avec des arguments déballés à partir d'une liste

Imprimer (numéros) # [2, 3, 4, 5,6]
```

Une liste ou un tuple peut également être déballé comme ceci:

```
Pays = ['Finlande', 'Suède', 'Norvège', 'Danemark', 'Island'] Fin, SW, Nor, * repos =
Pays Print (Fin, SW, Nor, rest) # Finland Sweden Norway ['Denmark', 'Island'] No
mbres = [1, 2, 3, 4, 5, 6, 7] Imprimer (un, milieu, dernier) # 1 [2, 3, 4, 5, 6] 7
```

Déballage des dictionnaires

```
Def Dontacking_Person_Info (Nom, Country, City, Age): Return f '{Name} vit dans {Cou ntry}, {City}. Il a {âge} an. '  DCT = {'Name': 'Asabeneh', 'Country': 'Finland', 'City': 'Hels inki', 'Age': 250} Print (Dentacking_Person_Info (** DCT)) # Asabeneh Live in Finland, Helsinki. Il a 250 ans.
```

Emballage

Parfois, nous ne savons jamais combien d'arguments doivent être transmis à une fonction Python. Nous pouvons utiliser la méthode d'emballage pour permettre à notre fonction de prendre un numéro illimité ou un nombre arbitraire d'arguments.

Listes d'emballage

```
def sum_all(*args):
    s = 0
    for i in args:
        s += i
    return s
print(sum_all(1, 2, 3))              # 6
print(sum_all(1, 2, 3, 4, 5, 6, 7)) # 28
```

Dictionnaires d'emballage

```
def packing_person_info (** kwargs): # Vérifiez le type de kwargs et il s'agit d'
un type de dict # print (type (kwargs)) # Impression des éléments de dictionnair
e pour la clé dans kwargs: print (f "{key} = {kwargs [key]}") return kwargs




print (packing_person_info (nom = "asabeneh", country = "Finland", ville =
"Helsinki", âge = 250)))

Nom = country asabeneh = Finland City = Helsinki Age = 250 {'name': 'Asabeneh', 'country': '
Finland', 'City': 'Helsinki', 'Age': 250}
```

Se propager en python

Comme dans JavaScript, la diffusion est possible dans Python. Vérifions-le dans un exem ple ci-dessous:

```
lst_one = [1, 2, 3] lst_two = [4, 5, 6, 7] lst = [0, * lst_one, * lst_two] print (lst) # [0, 1, 2,
3, 4, 5, 6, 7] country_lst_one = [" ',', ', nordeden' country_lst_two = ['Denmark', 'Island']
nordic_countries = [* country_lst_one, * country_lst_two] imprimer (nordic_countries) #
['Finland', 'Sweden', 'Norway', 'Denmark', 'Iceland']
```

Énumérer

Si nous sommes intéressés par un index d'une liste, nous utilisons la fonction intégrée *enumerate* pour obtenir l'index de chaque élément de la liste.

```
Pour l'index, élément en énumération ([20, 30, 40]): imprimer (index, élément) pou
r index, i en énumération (pays): print ('hi') si i == 'Finland': print ('le pays {i} a été
trouvé à index {index}')
```

La Finlande du pays a été trouvée à l'indice 1.

Fermeture éclair

Parfois, nous aimerions combiner des listes lors de la parole à travers eux. Voir l'exempl
e ci-dessous:

```
fruits = ['banane', 'orange', 'mango', 'citron', 'lime']
Légumes = ['Tomato', 'Potato', 'Cabbage', 'Onion', 'Carrot'] fruits_and_Veges = [] pour f, V dan
s Zip (fruits, légumes): Fruits_and_Veges.APPEND ({'Fruit': F, 'Veg': V})



print (fruits_and_veges)
```

```
[{'Fruit': 'banana', 'veg': 'tomato'}, {'fruit': 'orange', 'veg': 'pommes de terre'}, {'fruit': 'mango'
, 'veg': 'Cabbage'}, {'fruit': 'citron', 'veg': 'onon'}, {{'fruit' ':' lime " ": '}]]
```

⚽Vous êtes déterminé. Vous êtes à 17 étapes la tête jusqu'à la grandeur. Faites maintenant q
uelques exercices pour votre cerveau et vos muscles.

Exercices: Jour 17

1. Noms = ['Finlande', «Suède», «Norvège», «Danemark», «Islande», «Estonie», «Russie»]. Déballer les cinq premiers pays et les stocker dans une variable Nordic_Countries, Stor e Estonia et Russie en ES, et RU respectivement.

Félicitations!