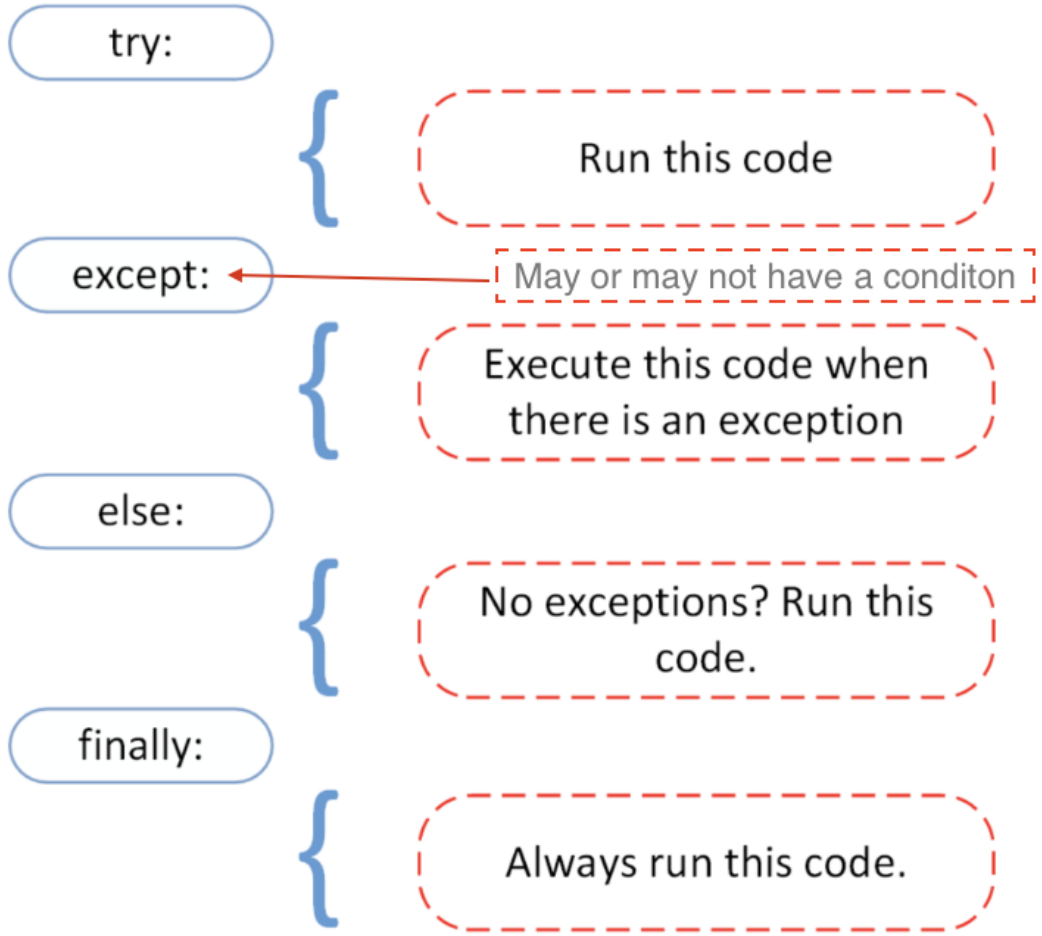
DAY-17 EXCEPTION HANDLING

Exception Handling

Python uses *try* and *except* to handle errors gracefully. A graceful exit (or graceful handling) of errors is a simple programming idiom - a program detects a serious error condition and "exits gracefully", in a controlled manner as a result. Often the program prints a descriptive error message to a terminal or log as part of the graceful exit, this makes our application more robust. The cause of an exception is often external to the program itself. An example of exceptions could be an incorrect input, wrong file name, unable to find a file, a malfunctioning IO device. Graceful handling of errors prevents our applications from crashing.

We have covered the different Python *error* types in the previous section. If we use *try* and *except* in our program, then it will not raise errors in those blocks.




```
try:
    code in this block if things go well
except:
    code in this block run if things go wrong
```

Example:

```
try:
    print(10 + '5')
except:
    print('Something went wrong')
```

In the example above the second operand is a string. We could change it to float or int to add it with the number to make it work. But without any changes, the second block, *except*, will be executed.

Example:

```
try:
    name = input('Enter your name:')
    year_born = input('Year you were born:')
    age = 2019 - year_born
    print(f'You are {name}. And your age is {age}.')
except:
    print('Something went wrong')
```

```
Something went wrong
```

In the above example, the exception block will run and we do not know exactly the problem. To analyze the problem, we can use the different error types with except.

In the following example, it will handle the error and will also tell us the kind of error raised.

```
try:
    name = input('Enter your name:')
    year_born = input('Year you were born:')
    age = 2019 - year_born
    print(f'You are {name}. And your age is {age}.')
except TypeError:
    print('Type error occured')
except ValueError:
    print('Value error occured')
except ZeroDivisionError:
    print('zero division error occured')

Enter your name:Asabeneh
Year you born:1920
Type error occured
```

In the code above the output is going to be *TypeError*. Now, let's add an additional block:

```
try:
    name = input('Enter your name:')
    year_born = input('Year you born:')
    age = 2019 - int(year_born)
    print(f'You are {name}. And your age is {age}.')
except TypeError:
    print('Type error occur')
except ValueError:
    print('Value error occur')
except ZeroDivisionError:
    print('zero division error occur')
else:
    print('I usually run with the try block')
finally:
    print('I alway run.')
```

```
Enter your name:Asabeneh
Year you born:1920
You are Asabeneh. And your age is 99.
I usually run with the try block
I alway run.
```

It is also shorten the above code as follows:

```
try:
    name = input('Enter your name:')
    year_born = input('Year you born:')
    age = 2019 - int(year_born)
    print(f'You are {name}. And your age is {age}.')
except Exception as e:
    print(e)
```

Packing and Unpacking Arguments in Python

We use two operators:

- * for tuples
- ** for dictionaries

Let us take as an example below. It takes only arguments but we have list. We can unpack the list and changes to argument.

Unpacking

Unpacking Lists

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

```
    return a + b + c + d + e

lst = [1, 2, 3, 4, 5]
print(sum_of_five_nums(lst)) # TypeError: sum_of_five_nums()
missing 4 required positional arguments: 'b', 'c', 'd', and
'e'
```

When we run the this code, it raises an error, because this function takes numbers (not a list) as arguments. Let us unpack/destructure the list.

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

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

We can also use unpacking in the range built-in function that expects a start and an end.

```
numbers = range(2, 7)  # normal call with separate arguments
print(list(numbers)) # [2, 3, 4, 5, 6]
args = [2, 7]
numbers = range(*args)  # call with arguments unpacked from a
list
print(numbers)      # [2, 3, 4, 5,6]
```

A list or a tuple can also be unpacked like this:

```
countries = ['Finland', 'Sweden', 'Norway', 'Denmark',
'Iceland']
fin, sw, nor, *rest = countries
print(fin, sw, nor, rest)   # Finland Sweden Norway
['Denmark', 'Iceland']
numbers = [1, 2, 3, 4, 5, 6, 7]
one, *middle, last = numbers
print(one, middle, last)      #  1 [2, 3, 4, 5, 6] 7
```

Unpacking Dictionaries

```
def unpacking_person_info(name, country, city, age):
    return f'{name} lives in {country}, {city}. He is {age}
year old.'
dct = {'name':'Asabeneh', 'country':'Finland',
'city':'Helsinki', 'age':250}
print(unpacking_person_info(**dct)) # Asabeneh lives in
Finland, Helsinki. He is 250 years old.
```

Packing

Sometimes we never know how many arguments need to be passed to a python function. We can use the packing method to allow our function to take unlimited number or arbitrary number of arguments.

Packing Lists

```
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
```

Packing Dictionaries

```
def packing_person_info(**kwargs):
    # check the type of kwargs and it is a dict type
    # print(type(kwargs))
    # Printing dictionary items
    for key in kwargs:
        print(f"{key} = {kwargs[key]}")
    return kwargs

print(packing_person_info(name="Asabeneh",
      country="Finland", city="Helsinki", age=250))

name = Asabeneh
country = Finland
city = Helsinki
age = 250
{'name': 'Asabeneh', 'country': 'Finland', 'city': 'Helsinki',
'age': 250}
```

Spreading in Python

Like in JavaScript, spreading is possible in Python. Let us check it in an example below:

```
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 = ['Finland', 'Sweden', 'Norway']
country_lst_two = ['Denmark', 'Iceland']
nordic_countries = [*country_lst_one, *country_lst_two]
print(nordic_countries)  # ['Finland', 'Sweden', 'Norway',
'Denmark', 'Iceland']
```

Enumerate

If we are interested in an index of a list, we use *enumerate* built-in function to get the index of each item in the list.

```
for index, item in enumerate([20, 30, 40]):
    print(index, item)

for index, i in enumerate(countries):
    print('hi')
    if i == 'Finland':
        print('The country {i} has been found at index
{index}')

The country Finland has been found at index 1.
```

Zip

Sometimes we would like to combine lists when looping through them. See the example below:

```
fruits = ['banana', 'orange', 'mango', 'lemon', 'lime']
vegetables = ['Tomato', 'Potato', 'Cabbage','Onion', 'Carrot']
fruits_and_veges = []
for f, v in zip(fruits, vegetables):
    fruits_and_veges.append({'fruit':f, 'veg':v})

print(fruits_and_veges)
```

```
[{'fruit': 'banana', 'veg': 'Tomato'}, {'fruit': 'orange',
'veg': 'Potato'}, {'fruit': 'mango', 'veg': 'Cabbage'},
{'fruit': 'lemon', 'veg': 'Onion'}, {'fruit': 'lime', 'veg':
'Carrot'}]
```

😳 You are determined. You are 17 steps a head to your way to greatness. Now do some exercises for your brain and muscles.

Exercises: Day 17

1. names = ['Finland', 'Sweden', 'Norway','Denmark','Iceland', 'Estonia','Russia']. Unpack the first five countries and store them in a variable nordic_countries, store Estonia and Russia in es, and ru respectively.

🎉 CONGRATULATIONS ! 🎉