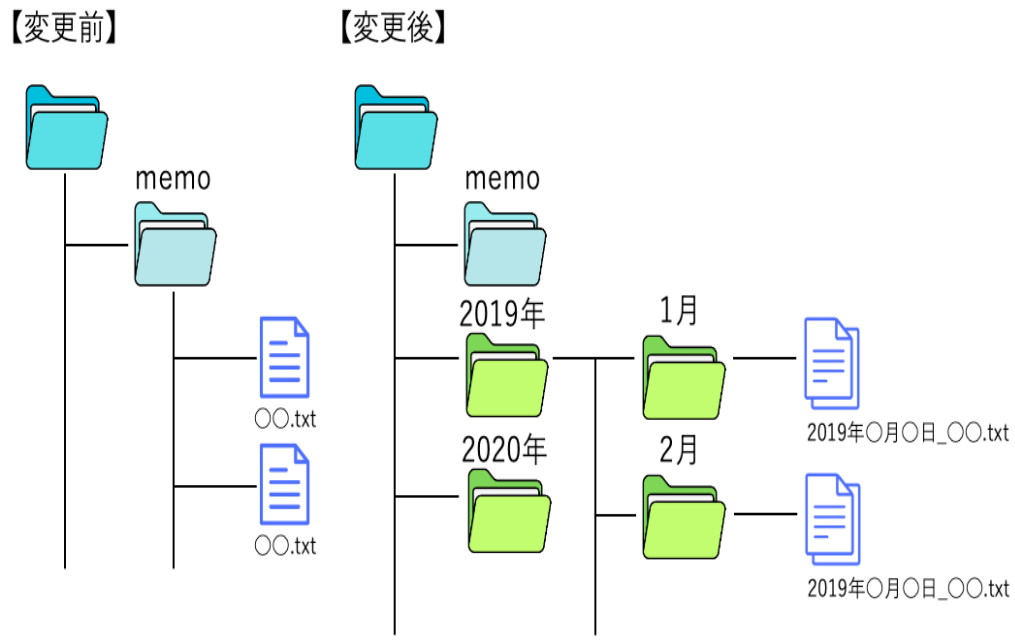
**業務効率化①「大量のファイルを一撃で整理しよう!」**

ここで取り組む課題としては、ファイル整理をやってみよう!これは Python を学ぶ上で初級中の初級であって、必須スキルだよ!
まずは、今回実施する流れを具体的に説明すると、
① 各個人のファイルを年・月ごとにフォルダに格納
② ファイル内にある日付と面談者名が入っている規則性を持ったファイル名に変える
この 2 点について作業をしていく。

ファイルが数個であれば手作業でいいけど、何千・何万というファイルを企業は取り扱っているため、プログラミング Python を活用して楽して整理しようということ!

このようにしていくよ!




プログラミングのコツとして「処理工程を分解!」してコーディングすることが重要!
実際の開発現場では、それじゃ!やること決まったしコーディングに取り掛かろう!ってことには絶対にならない!
コーディングに入るために現場のエンジニアはこの処理工程を分解の部分を一部として、「要件定義」としてユーザーストーリ（ペルソナの決定等々）・外部設計・内部設計などを細かく決めてからコーディングするんだけど、今回はその一部分の作業なので省略するね。

処理の流れとしては・・・
【パターン 1】
ステップ 1・・・memo フォルダ内にある各ファイルの作成日データを取得
ステップ 2・・・取得した作成日のデータを元に全ファイルの名前を変更
ステップ 3・・・取得した作成日のデータを元に各ファイルを任意のフォルダに移動

【パターン 2】
ステップ 1・・・memo フォルダ内にある各ファイルの作成日データを取得
ステップ 2・・・取得した作成日のデータを元に各ファイルを人のフォルダに移動
ステップ 3・・・フォルダ分けされた各ファイルの作成日データを取得
ステップ 4・・・取得した作成日のデータを元にフォルダ分けされた各ファイル名を変更

## 1. 準備

・デスクトップに「作業用」フォルダを作成
・file-org.ipynb ファイルで作業する
・その中に「memo」フォルダを作成
・ターミナルで jupyter-lab を起動

## 2. ファイルの作成日を取得

### 2.1　ファイルパスを変数に代入

#ユージ.txt のファイルパスを変数に格納

```
file_path = 'memo/ユージ.txt'
```

このコードは、ファイルパスを文字列(‘memo/ユージ.txt’)を file_path という変数へ格納し、文字列は「memo」フォルダの中の「ユージ.txt」ファイルだよ!という意味を表す。

### 2.2　os モジュールでファイルの作成日を取得

次に、ファイルの作成日を取得したいけど、変数に格納している file_path から取得できない。
file_path の中身は、ファイルを格納しているわけではなくて、文字列でファイルの位置を示しているだけであって、取得する場合には os モジュールという Python 標準の機能を使用する。

```
import os
```

※os は、オペレーティングシステムが提供している各種機能を扱えるようにするモジュール（モジュールは“ひとかたまりの機能を持つ部品（独立した単位）”です。

### 2.3 Mac でファイル情報を取得

Mac の場合は os の機能の stat()関数を使用し、ファイルパス(file_path)を指定することで、ファイル情報を詳細に取得することができる。
モジュール(os)内の関数を使用したいときは、モジュール名.関数名()をドット(.)で繋ぎ、()に変数を入れる。
なので今回は、os.stat(file_path)というコードになる。
関数が返す値を変数で受け取るには変数=関数()と記述し、これを代入という。

```
stat = os.stat(file_path)
print(stat)
```

os.stat(file_path)で取得した値を変数 stat に代入して、print()関数で出力して中身を確認してみよう。

出力結果

```
os.stat_result(st_mode=33204, st_ino=40821521, st_dev=16777233, st_nlink=1,
st_uid=501, st_gid=20, st_size=207, st_atime=1756019751,
st_mtime=1756019747, st_ctime=1756019749)
```

これはエラーではなくファイルの詳細情報が変数 stat の st_birthtime 属性(ファイルの作成日)が存在しているから。

ファイルの作成時刻を created_time という変数に代入して表示してみよう。

```
created_time = stat.st_birthtime
print(created_time)
```

出力結果

```
1582503600.0
```

この数値はタイムスタンプといい、ファイルの作成や更新といったイベント日時を記録している情報だけど、これじゃ何年何月何日なのかわからないから時間形式に変換して表示してみよう。

```
from datetime import datetime
```

datetime の datetime.fromtimestamp()関数を使用すると、データを変換してくれます。

```
Print(datetime.fromtimestamp(created_time))
```

出力結果

```
2020-02-24 09:20:00
```

出力は、2020 年 2 月 24 日に作成されたと確認できるよね。

では、変換後の値を再度、変数 created_time に代入しておこう。

```
created_time = datetime.fromtimestamp(created_time)
print(created_time)
```

出力結果

```
2020-02-24 09:20:00
```

### 2.4 datetime から「年・月・日」を取り出す

ファイル名の変更、フォルダの移動に必要な情報は年・月・日だけで、時間は必要がない。なので、created_time から年月日を個別に抽出してみよう。

まず、datetime 型は、年情報を year、月情報を month、日情報を day という属性を持つためそれぞれを取得してみる。
モジュールの関数記述方法と似ているけど、属性の場合は()は必要ない。

年情報を抽出

```
yeat = created_time.year
print(year)
```

出力結果

```
2020
```

月情報を抽出

```
month = created_time.month
print(month)
```

出力結果

```
2
```

日情報を抽出

```
day = created_time.day
print(day)
```

出力結果

```
24
```

これで、ファイルの作成日に関する情報が抽出し取得できた。

2.5 コードを１つのセルにまとめておこう

セルを全選択して、選択したセルを結合するでまとめておこう！

```
import os

file_path = 'memo/ユージ.txt'
stat = os.stat(file_path)
print(stat)

created_time = stat.st_birthtime
print(created_time)

from datetime import datetime
print(datetime.fromtimestamp(created_time))

created_time = datetime.fromtimestamp(created_time)
print(created_time)

year = created_time.year
print(year)

month = created_time.month
print(month)

day = created_time.day
print(day)
```

コードを整理しよう

```
import os
from datetime import datetime

file_path = 'memo/ユージ.txt'

stat = os.stat(file_path)

created_time = stat.st_birthtime

created_time = datetime.fromtimestamp(created_time)

year = created_time.year

month = created_time.month

day = created_time.day

print(year,month,day)
```

出力結果

```
2020 2 24za
```

ちゃんと出力されていればOK

こういう業務をする場合、最初は「1ファイル分の処理」だけをするようにしよう!
慣れてきても最初から複数のデータを扱わないことが整理されそして後から見返してもわかりやすいコードを書くことができる。
そして、複数ファイルを一気に処理するには、繰り返しの仕組みを使うことが不可欠であり、エラーにもなりやすいので、
① まずは1つのデータで処理を実装
② 問題がなかったら複数データを処理を実装

## 3. 抽出した作成日をもとにファイル名を変更してみよう

### 3.1 新しいファイル名にしてみよう

「〇年〇月〇日_個人情報_ユージ.txt」のような形式にしてみよう。
まずは、file_path から現在のファイル名「ユージ.txt」を取得して、前に取得した作成日の情報と組み合わせてみよう。変更後のファイル名は、変数 new_file_name に格納するようにしてみよう。

・file_path　memo/ユージ.txt
　ファイル名を抽出
・作成日と合体
　New_file_name　〇年〇月〇日_個人情報_ユージ.txt

ファイルパス=memo/ユージ.txt からファイル名=ユージ.txt だけを抽出して、文字列を分解する split()関数を使用する。
文字列.split(引数)を入力すると、引数で指定した値で文字列を分解してリストに格納することができる。
ることで分割してファイル名を抽出

・file_path.split( ‘/’ )[1]


```
file_path.split( ‘/’ )[1]
```

出力結果

```
’ユージ.txt’
```

### 3.2 抽出したファイル名は、変数 file_name に代入

・パスからファイル名を抽出

```
file_name = file_path.split( '/' )[1]
print(file_name)
```

出力結果

```
ユージ.txt
```

パスからファイル名を抽出できたので、変数の year・month・day と合体させて、新しいファイル名にしてみよう。
その際は、f 文字列(f-string=変数の埋め込みで特殊な文字列データで、f' ' の中に{変数}と記述することで代入ができる。)を使用して変数に埋め込む。

```
new_file_name = f'{year}年{month:02}月{day:02}日_{file_name}'
print(new_file_name)
```

出力結果

```
2020 年 02 月 24 日_ユージ.txt
```

### 3.3 ファイル名を変更しよう

ファイル名を変更するためには、os の rename()関数を使用する。
os.rename(変更前,変更後)という形で引数を指定する。
ここで注意!
引数に変更前と変更後のファイルパスを指定することが大前提!ファイル名を指定してしまうとエラーが出る又は意図せぬ場所にコピーされる危険があるよ!
なので変更前のファイルパスは file_path でいい!新しいファイルパスは、変更前のフォルダ「memo」にして、ファイル名だけを変更しよう!なので「memo/新しいファイル名」として、f 文字列で定義するので、「f'memo/{new_file_name}'」というコードになる。

```
os.rename(file_path, f'memo/{new_file_name}')
```

ファイル名が変更されたことを確認し、コードをまとめておこう

### 3.5 作成日をもとにファイルをフォルダ分けしよう!

同じ名前のフォルダが存在するのかしないのかで、処理の流れを変えていく。

では、処理の全体像をイメージしてみよう!

今回のプログラムは、作成日が 2021 年 4 月 16 日のファイルがあるとすれば、「2021 年」フォルダの中の「4 月」フォルダに格納されるように処理したらかなり整理されるよね!
仮にもう「2021 年」というフォルダや「4 月」というフォルダが作成しているなら作成する必要はないけど、まだ作成していないのであれば自動で作成してファイルをそのフォルダに移動する必要があるよね!

この処理をしていくためには、分岐処理 if 文を使用する。
ある条件を満たしていなければ①の処理、そうでなければ②の処理を行う!

**「○年」「○月」のフォルダ**

| ① 存在する | ② 存在しない |
|---|---|
| ・「○年」「○月」フォルダにファイルを格納 | ・「○年」「○月」フォルダを作成<br>・「○年」「○月」フォルダにファイルを格納 |

んじゃ指定するフォルダがあるかないかを確認してみよう!

まずはテスト用に「2020 年」というフォルダだけを手動で作成しておき、ファイルを振り分ける「interview」フォルダの中に作成しておこう!

では、フォルダが存在するか否かを確認するためのコードは、os の path モジュールにある exists()関数をしようして確認する。
引数にはフォルダのパスを指定するので、os.path.exists(フォルダのパス)と記述すると、存在する場合は True、ない場合は False を返してくれるよ。

```
print(os.path.exists('interview'))
print(os.path.exists('2020年'))
print(os.path.exists('interview/2020年'))
print(os.path.exists('interview/2021 年'))
```

出力結果

```
True
False
True
False
```

要はこのコード確認するのは、存在するか存在しないかで True/False を返してくれた。

### 3.6 これを活用して if 文で条件分岐を指定こう！

if 文で「存在しない場合」はフォルダを作成する！
if 文は、条件を満たすときだけ指定のコードを実行する命令文で、開発において必ず絶対使用するよ！

ここで復習 if 文の書き方

```
if 条件式:
    条件式が True の時の処理
else:
    条件式が False の時の処理
```

・not 演算子で反転！
試しに「2021 年」フォルダを作って同名のフォルダが存在するのかチェックして、存在しない場合だけフォルダを作成するコードを書こう！

```
if not os.path.exists('interview/2021年/4月'):
    os.makedirs('interview/2021 年/4 月')
```

・ポイント①
if 文の条件式に not 演算子を使用している。
これは、引数に指定したフォルダが存在しない場合 os.path.exists()は False を返す。
このままならフォルダが存在しないときは、次のコードが処理されない命令になってしまうよね。
なので、条件式の前に not をつけることで、false を反転させ、True にしている。これによってフォルダが存在しない場合、次のコードを実行させる、という条件を指定できるようになった。

ちょっとイメージしづらいかもなので、os.path.exists()関数で存在しない test フォルダを確認して、そのまま出力させて not 演算子で値を反転させてから出力するコード書いてみよう。

```
print(os.path.exists('test'))
print(not os.path.exists('test'))

if not os.path.exists('test'):
    print('testフォルダは存在しません')
```

出力結果

```
False
True
test フォルダは存在しません
```

１行目は false を出力しているのに対して、２行目は not をつけたコードなので True が出力されているよね。

os.path.exists()が返す値(false)を反転させることで、次の命令分が実行できるようになっている。
このように、条件式が false を返すとき条件を満たす if 文を書くには、not 演算子が不可欠である。

・ポイント②
os.makedirs()でフォルダをまとめて作成!

フォルダを作成する関数としてos.makedirs()関数を使用している!
この関数は、引数にフォルダパスを指定すると複数の階層でも一気に作成してくれる。
今回の引数は、「’interview/2021年/4月’」と指定しているよね。
Interviewの中に「2021年」フォルダを作成して、2021年の中に「4月」フォルダが作成されているのを確認しよう。

ただ、今のコードでは汎用性が低いため、汎用性が高いコードに変更してみよう!

要は、「2021年」フォルダの中に「4月」フォルダしか存在確認をしてくれないコードなため、汎用性が低いコードとなってしまっている。
なので、ファイル毎に適切なフォルダを作成できるようにしよう!

```
new_dir = f’interview/{year}年/{month}月’
if not os.path.exists(new_dir):
    os.makedirs(new_dir)
```

このコードは、変数とf文字列を組み合わせてコピー先のフォルダ名をf’interview/{year}年/{month}月’として、変数new_dirに代入している。そして次のif文で、このnew_dirの存在を確認して、存在しなければ作成する。これによって、yearやmonthの値に応じたフォルダ名が変動するコードとなる。

### 3.7 目的のフォルダにファイルを移動してみよう!

shutil.moveでファイルを移動

では、作成したフォルダにファイルを移動してみよう。
ファイルの伊藤にはshutilモジュールのmove()関数を使用するよ。
この関数は二つの引数を取って、shutil.move(変更前、変更後)と記述する。
※shutilとは、ファイルやフォルダの操作に関する様々な機能を提供してくれる。

ここで注意、プログラミングでファイルを操作すると、通常ドラック&ドロップとか、間違えたからもとに戻そうとか、気軽にできない!なのでshutil.move()を実行するときは、ちゃんと移動後のファイルパスがちゃんと指定できているかを確認しよう!

移動させるファイルパスは長いためdest変数に代入するよ。

```
print(f'memo/{new_file_name}')
dest = f'interview/{year}年/{month}月/{new_file_name}'
print(dest)
```

出力結果

```
memo/2020年07月16日_杏.txt
interview/2020年/7月/2020年07月16日_杏.txt
```

移動前と移動後のファイルパスが想定通りの出力結果になっているよね。

では、リネームしたファイルを新しいパスに移動してみよう。

```
import shutil
shutil.move(f'memo/{new_file_name}', dest)
```

出力結果

```
'interview/2020年/7月/2020年07月16日_杏.txt'
```

では、移動したかを確認してみよう！

移動が確認できたら、コードを一つにまとめておこう!

```
import os
from datetime import datetime
import shutil

file_path = 'memo/岡田准一.txt'

stat = os.stat(file_path)

created_time = stat.st_birthtime

created_time = datetime.fromtimestamp(created_time)

year = created_time.year

month = created_time.month

day = created_time.day

file_name = file_path.split('/')[1]
new_file_name = f'{year}年{month:02}月{day:02}日_{file_name}'
os.rename(file_path, f'memo/{new_file_name}')

new_dir = f'interview/{year}年/{month}月'
if not os.path.exists(new_dir):
    os.makedirs(new_dir)

dest = f'interview/{year}年/{month}月/{new_file_name}'

shutil.move(f'memo/{new_file_name}', dest)
```

## 4. 全てのファイルを処理しよう！

### 4.1 Glob()関数でファイルパスを一気に取得!

では、最後の行程に入ろう、これまでは一件のみのコードだったけど、これを一気に処理していく。
そこで使用する関数が、glob()関数で、簡単にファイルパスをまとめて取得できるようになるよ。

glob モジュールにある glob()関数をインポートしよう。

```
from glob import glob
```

glob()は、指定したパターンにマッチするファイルを一括で取得して、リストとして返す関数だよ。

引数には、取得したいフォルダ・ファイルのパスを文字列で指定するけど、このときはパス内にワイルドカードを埋め込む。
※ワイルドカードとは、不特定の文字や文字列であることを示す記号のこと。

検索でよく用いられるのが、「*」や「?」だけど使用回数が多いのは「*」であり、「*」は 0 文字以上の任意の文字列を表すワイルドカードとなる。

要は、0 文字以上なら、どのような文字に対しても使用できるよってこと。

じゃ実行してみよう！

```
glob('*')
```

出力結果

```
'interview',
'ファイル整理.ipynb',
'memo',
```

ワイルドカードには、任意の文字列を組み合わせることができて、'*_ipynb'と拡張子を指定すると ipynb ファイルだけを取得することができる。

```
glob('*.ipynb')
```

出力結果

```
'ファイル整理.ipynb'
```

今回は、memo フォルダ内にある拡張子.txt だけを取得したいので、ワイルドカードを組み合わせて、' memo/*.txt と引数を指定する。
ここでは先頭の 5 つだけを表示してみるよ。

```
file_paths = glob('memo/*.txt')
print(file_paths[:5])
```

出力結果

```
['memo/松嶋菜々子.txt', 'memo/平手友梨奈.txt', 'memo/清原果耶.txt', 'memo/福原遥.txt', 'memo/仲間由紀恵.txt']
```

こう言ったように glob()関数とワイルドカードを組み合わせて結構使用するから覚えておこう。

### 4.2 ファイルの指定を glob()関数に置き換えよう！

今のコード組み合わせれば、一気にいけるんだけど、整理しながら順を追って完成させていこう！

まず、先頭にある変数 file_path は、memo フォルダ内にある 1 つのファイルのパスを代入してたよね!
要は、ファイルパスを一つだけ代入すればいいから、インデックス番号[0]で指定するコードに書き換えれる。

```
file_path = file_paths[0]
print(file_path)
```

出力結果

```
memo/松嶋菜々子.txt
```

というように次のようなコードに書き換えれるよ!

```
import os
import datetime
import shutil

file_paths = glob('memo/*.txt')
file_path = file_paths[0]

print(file_path)

stat = os.stat(file_path)
created_time = stat.st_birthtime

created_time = datetime.datetime.fromtimestamp(created_time)

year = created_time.year
month = created_time.month
day = created_time.day

file_name = os.path.split('/')[1]
new_file_name = f'{year}年{month}月{day}日_{file_name}'
os.rename(file_path, f'memo/{new_file_name}')

new_dir = f'interview/{year}年/{month}月'
if not os.path.exists(new_dir):
    os.makedirs(new_dir)

dest = f'interview/{year}年/{month}月/{new_file_name}'
shutil.copy(f'memo/{new_file_name}', dest)
```

出力結果

```
memo/平手友梨奈.txt
'interview/2020年/3月/2020年3月16日_'
```

一つのファイルに対して行なっていた処理を for 文を用いて全部適用してみよう!

### 4.3 for 文のおさらい

for 文は、同じ処理を繰り返したいときに使用する構文だったよね!
今回のようなリスト内の各要素に同じ処理をしたい場合、for 文が活かせれる!

```
for 変数i in リストの変数:
    繰り返しの処理
```

in の後にリストの変数を置くと、そのリスト内の要素を一つずつ変数 i に代入して、次へ次へと繰り返し処理を実行する。

**for 文で全部処理してみよう!**

変更するコードは 2 箇所、file_path=file_paths[0]を for file_path in file_paths:に変えて、for 文をコーディングする。

```
import os
import datetime
import shutil
from glob import glob

file_paths = glob('memo/*.txt')

for file_path in file_paths:
    stat = os.stat(file_path)
    created_time = stat.st_birthtime

    created_time = datetime.datetime.fromtimestamp(created_time)

    year = created_time.year
    month = created_time.month
    day = created_time.day

    file_name = os.path.split(file_path)[1]
    new_file_name = f'{year}年{month}月{day}日_{file_name}'
    os.rename(file_path, f'memo/{new_file_name}')

    new_dir = f'interview/{year}年/{month}月'
    if not os.path.exists(new_dir):
        os.makedirs(new_dir)

    dest = f'interview/{year}年/{month}月/{new_file_name}'
        shutil.copy(f'memo/{new_file_name}', dest)
```

コードが完成したので実行して確認してみよう！

ちゃんと整理されていれば完成だよ！

Python で業務効率化を実装することで重要なのは・・・
① 作業行程を分解して、一つずつ確認しながら進めていくこと！
② 複数ファイルを一気に処理しようとしないで、1 ファイルから確実に実行できていることを確認しながら実装すること！

この２つを守ることで、意図しないエラーや後戻りのない無駄なコードを記述しなくても済むよ！