PythonBasicB

ファイルの取り扱い

```
//ファイルを開いて中身を読み取る
with open ("sample.txt","r",encoding="utf-8") as file:
    data = file.read()
    print(data)
```

mode	説明	例
r	読み込み	
w	書き込み(新規作成)	
a	追記	
r+	読み書き	
w+	読み書き(新規作成)	
a+	追記・読み書き	
t	テキストモード	
b	バイナリモード	画像など

```
//ファイルを新規作成して、文字列を書き込んで保存
with open ("data.txt", "w", encoding="utf-8") as file:
    file.write("Python\n") #\nは改行を意味します
    file.write("保存")
```

- with 文を使用しない場合はclose()メソッドでファイルを閉じる必要があります。

ディレクトリからファイルを取得する

globライブラリを使用する場合

```
import glob

files = glob.glob("./sample_dir/*")
print(files)
for file in files:
    print(file)
```

osライブラリを使用する場合

```
import os

files = os.listdir("./sample_dir")
print(files)
for file in files:
    print(file)
```

練習問題：ファイルに保存する機能を使って、アプリケーションを永続化する

クラスとオブジェクト

クラスはオブジェクトの設計図であり、保持するデータ(変数)と振る舞い(関数・メソッド)をひとまとめにしたもの。

```
# クラス定義
class Ticket:
    event_name = "肥後橋夏フェス"
    sheet_no = "A200"
    price = 5000

    def show_detail(self):
        return f"公演名:{self.event_name} 座席番号:{self.sheet_no} チェット代
金:{self.price}"

# 変数tiにクラス設計図から生成したオブジェクトを代入(インスタンス化)
ti = Ticket()
# インスタンスのメソッドを実行
print(ti.show_detail())
```

コンストラクター

クラスをインスタンス化する時にだけ自動的に実行されるメソッドをコンストラクタという。
逆にインスタンスが消滅するときにだけ自動的に実行されるメソッドをデストラクタという。

```
class Ticket:
    # コンストラクタ
    def __init__(self, event_name,sheet_no,price):
        self.event_name = event_name
        self.sheet_no = sheet_no
        self.price = price

    def show_detail(self):
        return f"公演名:{self.event_name} 座席番号:{self.sheet_no} チェット代
金:{self.price}"
```

継承

クラス(オブジェクトの設計図)からその属性を受け継ぎ、派生したオブジェクト作るとこを継承という。

```
# Ticketを継承してVipTicket(招待チケット)を作る
class VipTicket(Ticket):
    # コンストラクタ
    def __init__(self, event_name,sheet_no):
        self.event_name = event_name
        self.sheet_no = sheet_no
        self.price = 0

vti = VipTicket("肥後橋夏フェス","Z001")
# Ticketを継承しているため、show_detailメソッドはVipTicketクラスに定義しなくても備えている
print(vti.show_detail())
```

メソッドのオーバーライド

```
class VipTicket(Ticket):
    # コンストラクタ
    def __init__(self, event_name,sheet_no):
        self.event_name = event_name
        self.sheet_no = sheet_no
        self.price = 0
    # メソッドを上書き(オーバーライド)
    def show_detail(self):
        return f"公演名:{self.event_name} 座席番号:{self.sheet_no} チェット代金:招待"
# 継承クラスのインスタンス化
vti = VipTicket("肥後橋夏フェス","Z001")
# 継承クラスのメソッド実行
print(vti.show_detail())
# 親クラスのメソッド実行
print(super(VipTicket,vti).show_detail())
```

アルゴリズム

調理に置き換えると.... 文法は「食材を切断する道具を包丁という」や「包丁の握り方」など調理道具の名称や使い方です。
アルゴリズムは「出汁の取り方」「肉じゃがの作り方」などレシピに該当します。
また「肉じゃがの作り方」が調理者や材料により複数存在するように同じ目的のアルゴリズムも複数存在します。

基本のアルゴリズム

例：1からnまで足し合わせる

```
# ひたすら足すアルゴリズム(簡単だけど100とか1000になると大変)
total = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10
```

```
# 繰返しを利用するアルゴリズム
n = 10
total = 0
for i in range(1,n+1):
    total += i
```

別の解釈をすると
1 + 10 = 11
2 + 9 = 11 3 + 8 = 11 一番先頭と一番最後の数 二番目と後ろから二番目の数 三番目と後ろから三番目の数 を加えてもすべて11となります。

```
# 繰返しを使わず、計算で求めるアルゴリズム
total = (1 + n) * n / 2
```

と複数の計算方法があります。 この計算方法がアルゴリズムです。

料理のレシピも複数ありますが、正しいかどうかは目的で異なります。

- 調理時間を短くしたい
- 材料費、光熱費を安くしたい
- 味を最優先

プログラムのアルゴリズムも目的で正解は異なります。

- 計算が早く終わる
- 修正が容易

素数を求める

素数とは1とその数以外に約数を持たない2以上の自然数です。
2,5,7,11,13...と無数に存在します。
素数の求め方にも複数の方法(アルゴリズム)があります。
プログラム初心者向けに繰返しと条件分岐を使って求める方法を学びます。
2 から 100 までの自然数nを 2 から n/2 まで順番に割って割り切れなければ素数と判断します。n/2までとするのはn/2より大きな数では割れないからです。
例) 50 は 25より大きな数字では割れません。

n!と再帰関数

nの階乗(1 x 2 x 3 x 4 x 5 .... x n)をn!と表します。 n! の計算方法(アルゴリズム)には繰返しを使う方法と再起関数を使うほうがあります。

```
# 繰返しを使う方法
n = 10
n = 10
total = n
```

```
for i in range(n-1, 0, -1):
    total = total * i
print(total)
```

再起とは自分自身を実行中で呼び出すプログラミング手法です。

```
def factorial(n):
    if n = 0:
        return 1 # 0!は1のため
    return n * factorial(n - 1)
```

Excelの操作

ライブラリのインストール

コンソールorターミナル画面にて

```
pip install openpyxl
```

ライブラリのインポート

```
import openpyxl
```

ファイルの操作

新しいワークブックを作成してsample.xlsxとして保存

```
workbook = openpyxl.Workbook()
workbook.save("sample.xlsx")
```

既存のファイルnumber.xlsxを編集して保存

```
workbook = openpyxl.load_workbook('number.xlsx')
# 編集作業をする
workbook.save('number.xlsx')
```

ワークシートの操作

ワークシートを選択

```
```

```
# シート名を指定
worksheet = workbook["Sheet"]
# アクティブなシートを選択
worksheet = workbook.active
# シート名の一覧を取得
workbook.sheetnames
```

セルの操作

```
# A1セルを取得
value = worksheet['A1'].value
value = worksheet.cell(row=1,column=1).value
# A1セルに文字列を書き込む
worksheet['A1'].value = "創造社デザイン専門学校"
worksheet.cell(row=1,column=1).value = "Sozosha"
```

重要メソッド

```
# シートの最大行(最終データが記載されている行)
maxrow = worksheet.max_row
# シートの最大列(最終データが記載されている列)
maxcol = worksheet.max_column
# シートの最小行(最初のデータが記載されている行)
minrow = worksheet.min_row
# シートの最大列(最終データが記載されている列)
minrcol = worksheet.min_column
```

練習問題

```
#下記の辞書型データを科目をA列、成績をB列に書き込み、exam.xlsxとして保存します。
exam = {"英語":80, "国語":78, "数学":65, "日本史":55, "生物":90}
#新規excelファイル作成
workbook = openpyxl.Workbook()
worksheet = workbook.active

#科目の一覧
subjects = exam.keys()
for index, subject in enumerate(subjects):
    worksheet.cell(row=index+1,column=1).value = subject

#成績の一覧
scores = exam.values()
for index, score in enumerate(scores):
    worksheet.cell(row=index+1,column=2).value = score

workbook.save('exam.xlsx')
```

```
# 作成したexam.xlsxを開き、B10セルに成績の合計、B11セルに平均を計算する
workbook = openpyxl.load_workbook('exam.xlsx')
worksheet = workbook.active

#データを読み取る
examdata = []
for i in range(1,worksheet.max_row+1):
    examdata.append(worksheet.cell(row=i,column=2).value)

worksheet["A10"].value = "合計"
worksheet["B10"].value = sum(examdata)
worksheet["A11"].value = "平均"
worksheet["B11"].value = sum(examdata)/len(examdata)

workbook.save('exam.xlsx')
```

課題イメージ