memo

プログラミング演習 I

授業開始までしばらくお待ちください。

授業前にこれを見ている人へ。

- 内容を先に読んで予習しておくことは大いに結構なことだと思います。ぜひ予習してください。
- 内容は随時更新しています。授業前に再度最新版を確認してください。
- 課題の提出 (Google Forms によるもの) は **授業当日の指示があるまで行わないように** してください。

自習教材その 1: Paiza ラーニング

1. ブラウザで `https://paiza.jp/works` を開く。
2. 【新規登録】をクリック。
3. 【Google で登録】→《アカウントの選択》で**工大の組織アカウントを選ぶ**。→ 認証を行う。
4. (右上の)【ユーザー】→【クーポンコード入力】と進みクーポンコードを入力する。
5. 【講座一覧】→【Python3】→【Python 体験編】と進み講座を始める。

自習教材その 2: prog-8

1. ブラウザで `https://prog-8.com` を開く。
2. 無料会員登録 をクリック。
3. 【Google で続ける】→《アカウントの選択》で**工大の組織アカウントを選ぶ**。
4. 【コース一覧】→【Command Line】と進みレッスンを始める。

環境構築状況の確認 (わからなければ SA に助けてもらう。)

Windows PC の人は以下をチェック

- Debian アプリを起動して、(設定をいじってなければ)『**自分で決めたユーザ名 @ 何か文字列: $**』と表示される。
- Debian の画面で **python3 --version** と入力すると**Python 3.??.?**みたいな表示が出る。
- スタートボタン →(Windows 11 の人はすべてのアプリ)→「V」欄 →Visual Studio Code が起動できる。

Mac の人は以下をチェック

- ターミナルを開き、**type python3** と入力し、返ってきた文字列の中に **homebrew** もしくは **local** が含まれる。
- LaunchPad もしくは Finder あるいはターミナルから VSCode が起動できる。

おまけ

- **Windows**
  Microsoft 社製の OS(ソフトウェアの一種)。**パソコンのことではない。**
- **Windows PC**
  OS として Windows が動いているパソコンのこと。ただし、Windows で使うことを想定した PC に、Windows 以外の OS(Linux 等) を入れている場合も “Windows PC” と言うことが (変だけれど) ふつうにある。
- **Mac**
  Apple 社製のパソコン。ソフトウェアではなく、**パソコンのこと**。M は大文字。昔は Macintosh いう製品名だったが、今は Mac。MacBook は Mac の一種。
- **macOS**
  Mac で使う OS(ソフトウェアの一種)。Windows と同じカテゴリの言葉。mac は小文字で OS は大文字。間にスペースを入れない。Windows が動くパソコンを “Windows PC” と言うが、macOS が動くパソコンを “macOS PC” と言うことは（理屈の上では正しいはずなのだけど）まずない。
- **PC**
  広義には (Mac も含めた) パソコン全般を、狭義には Windows PC を表す言葉。どっちを指すかは空気と文脈から読み解く必要がある。前者は一般名詞の “personal computer” の略、後者は’80 年代の “IBM PC” という製品名が由来。

これから最大 1 時間程度、

- 前のページのチェックが未完の人は、(SA や教員に手伝ってもらいながら) 環境構築を終わらせる。(終わったら prog-8 の演習の続きをやる。)
- そうでない人は prog-8 の演習の続きをやる。(Paiza は動画で音が出るので授業中はやらないこと。)

プログラミング演習 I
Exercise on Programming I

本スライドは Teams で配布しています。

小林裕之・瀬尾昌孝
prog1@oitech.ddns.net

大阪工業大学 RD 学部システムデザイン工学科
OSAKA INSTITUTE OF TECHNOLOGY

1 of 14

a LaTeX + Beamer slideshow

この授業でやること

- プログラミングの基礎・基本的な概念を演習を通じて学ぶ。
- プログラミング言語として Python（パイソン）を使う（が、Python であることはあまり重要ではない）。

評価方法

『演習』の授業は、出席して、授業に参加すること自体がとても重要です。

- **演習出席点 =3 点/回** (30 分以内の遅刻 =2 点, 途中無断退出 =0 点。出席管理システムで採点するので必ず学生証を通すこと。学生証を忘れたら**授業中に SA もしくは教員に申し出ること。**)
- **レポート点 =3 点/回**
- **期末演習点 =16 点**

以下はその都度適宜減点:
動画・音楽の視聴, イヤフォン等の使用 (聴覚の補助等を除く), ゲーム, 指定席以外への移動, 食事, 睡眠, その他常識の範囲で明らかに演習の授業を受けることに対して不適当な行為全般

公共交通機関の遅れで遅刻した場合

- 工大の Google アカウントにログインした状態で、
- ← この Google Forms から遅延証明を提出し、
- **prog1@oitech.ddns.net にその旨を連絡**する。

https://forms.gle/MWndV7k3eqTbS8Ly9

プログラミング的思考

『プログラミング的』思考

7					5	9		
	2			9			3	
		3	2		6			5
1	3					2		
		5				6		
		4					7	8
4			5		2	7		
	9			6			4	
		7	1					3

『プログラミング的』思考

どうやれば**確実に**解ける？

決してバカにしてはいけない解法

ナンプレの解法

問題点:

決してバカにしてはいけない解法

ナンプレの解法

すべての空きマスに 1〜9 すべての組み合わせを入れ、全部チェックする。

問題点:

決してバカにしてはいけない解法

ナンプレの解法

すべての空きマスに 1〜9 すべての組み合わせを入れ、全部チェックする。

問題点: たぶん時間がかかりすぎる。前ページの例題の場合、空きマスが 53 個なので、ぜんぶで 9^{53} 回のナンプレチェック作業が必要。

決してバカにしてはいけない解法

ナンプレの解法

すべての空きマスに 1〜9 すべての組み合わせを入れ、全部チェックする。

問題点: たぶん時間がかかりすぎる。前ページの例題の場合、空きマスが 53 個なので、ぜんぶで 9^{53} 回のナンプレチェック作業が必要。

毎秒 1 億個チェックしたとしても 1×10^{27} 億年以上かかる。

決してバカにしてはいけない解法

ナンプレの解法

すべての空きマスに 1〜9 すべての組み合わせを入れ、全部チェックする。

問題点: たぶん時間がかかりすぎる。前ページの例題の場合、空きマスが 53 個なので、ぜんぶで 9^{53} 回のナンプレチェック作業が必要。

毎秒 1 億個チェックしたとしても 1×10^{27} 億年以上かかる。

でも、少なくとも誰でも、確実に解ける方法を示せたことは重要。

たった 3 ステップで**誰でも、確実に、効率的に**解ける！

秘伝《ナンプレ q の解》 手順書

$q =$

7					5	9		
	2			9			3	
		3	2		6			5
1	3					2		
		5				6		
		4					7	8
4			5		2	7		
	9			6			4	
		7	1					3

1. q が
2. q が
3. q の

たった 3 ステップで**誰でも、確実に、効率的に**解ける！

秘伝《ナンプレ q の解》 手順書

$q =$

7					5	9		
	2			9			3	
		3	2		6			5
1	3					2		
		5				6		
		4					7	8
4			5		2	7		
	9			6			4	
		7	1					3

1. q が間違いだったら**解なし**。
2. q が
3. q の

たった 3 ステップで**誰でも、確実に、効率的に**解ける！

秘伝《ナンプレ q の解》 手順書

$q =$

7					5	9		
	2			9			3	
		3	2		6			5
1	3					2		
		5				6		
		4					7	8
4			5		2	7		
	9			6			4	
		7	1					3

1. q が間違いだったら**解なし**。
2. q が全マス埋まっていればq **が解**。
3. q の

たった 3 ステップで**誰でも、確実に、効率的に**解ける！

秘伝《ナンプレ q の解》 手順書

$q =$

7					5	9		
	2			9			3	
		3	2		6			5
1	3					2		
		5				6		
		4					7	8
4			5		2	7		
	9			6			4	
		7	1					3

1. q が間違いだったら**解なし**。
2. q が全マス埋まっていれば**q が解**。
3. q の最初の空きマスに 1 から 9 まで順に埋めた問題 $q_1, q_2, \cdots, q_9$ を作成し、それぞれについて**《ナンプレ q_i の解》**を求める。

たった 3 ステップで**誰でも、確実に、効率的に**解ける！

秘伝《ナンプレ q の解》 手順書

$q =$

7					5	9		
				9			3	
								5
	3					2		
		5				6		
		4					7	8
4			5		2	7		
	9			6			4	
		7	1					3

$1, 2, \cdots, 9$

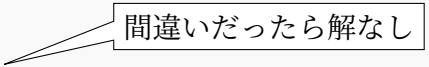
1. q が間違いだったら**解なし**。
2. q が全マス埋まっていれば**q が解**。
3. q の最初の空きマスに 1 から 9 まで順に埋めた問題 $q_1, q_2, \cdots, q_9$ を作成し、それぞれについて**《ナンプレ q_i の解》**を求める。

『プログラミング的思考』から本当の『プログラム』へ

```
solveSdk q =

  if chkSdk q == False then []

  else if zp == 81 then q

  else concatMap (\i -> let q' = (take zp q) ++ [i] ++ drop (zp + 1) q

                        in  solveSdk q') [1..9]

  where zp = index q 0
```

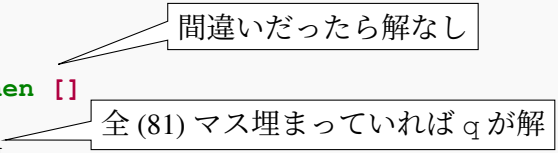
『プログラミング的思考』から本当の『プログラム』へ

```
solveSdk q =
  if chkSdk q == False then []
  else if zp == 81 then q
  else concatMap (\i -> let q' = (take zp q) ++ [i] ++ drop (zp + 1) q
                        in  solveSdk q') [1..9]
  where zp = index q 0
```



『プログラミング的思考』から本当の『プログラム』へ

```
solveSdk q =
  if chkSdk q == False then []
  else if zp == 81 then q
  else concatMap (\i -> let q' = (take zp q) ++ [i] ++ drop (zp + 1) q
                        in  solveSdk q') [1..9]
  where zp = index q 0
```



『プログラミング的思考』から本当の『プログラム』へ

```
solveSdk q =

  if chkSdk q == False then []

  else if zp == 81 then q

  else concatMap (\i -> let q' = (take zp q) ++ [i] ++ drop (zp + 1) q

                        in  solveSdk q') [1..9]

  where zp = index q 0
```

間違いだったら解なし

全 (81) マス埋まっていれば q が解

最初の空きマスに順に数字を入れて q' とし、q' の解を求める

その他少々加えて完成！(言語は Haskell)

```
q0 = [7, 0, 0, 0, 0, 5, 9, 0, 0,
      0, 2, 0, 0, 9, 0, 0, 3, 0,
      0, 0, 3, 2, 0, 6, 0, 0, 5,
      1, 3, 0, 0, 0, 0, 2, 0, 0,
      0, 0, 5, 0, 0, 0, 6, 0, 0,
      0, 0, 4, 0, 0, 0, 0, 7, 8,
      4, 0, 0, 5, 0, 2, 7, 0, 0,
      0, 9, 0, 0, 6, 0, 0, 4, 0,
      0, 0, 7, 1, 0, 0, 0, 0, 3]
main = mapM_ (\i -> print (take 9 (drop i ans))) [0,9..72] where ans = solveSdk q0
chkDup a = all (\i -> length (filter (== i) a) <= 1) [1..9]
chkSdk x =
  all (\i -> chkDup (take 9 (drop i x))) [0,9..72] &&
  all (\i -> chkDup (map (\j -> x!!(i + j)) [0,9..72])) [0..8] &&
  all (\i -> chkDup (map (\j -> x!!(i + j)) box)) tl
  where
    box = [0, 1, 2, 9, 10, 11, 18, 19, 20]
    tl = [0, 3, 6, 27, 30, 33, 54, 57, 60]
index [] _ = 0
index (x:xs) tgt = if x == tgt then 0 else 1 + index xs tgt
--
solveSdk q = if chkSdk q == False then []
             else if zp == 81 then q
             else concatMap (\i -> let q' = (take zp q) ++ [i] ++ drop (zp + 1) q
                                   in  solveSdk q') [1..9]
  where zp = index q 0
```

https://paiza.io/projects/xCJ5haGfhrTvbo4zw3GHzA

同じものを別言語 (この授業でやる Python) で

```
q0 = [7, 0, 0, 0, 0, 5, 9, 0, 0,
      0, 2, 0, 0, 9, 0, 0, 3, 0,
      0, 0, 3, 2, 0, 6, 0, 0, 5,
      1, 3, 0, 0, 0, 0, 2, 0, 0,
      0, 0, 5, 0, 0, 0, 6, 0, 0,
      0, 0, 4, 0, 0, 0, 0, 7, 8,
      4, 0, 0, 5, 0, 2, 7, 0, 0,
      0, 9, 0, 0, 6, 0, 0, 4, 0,
      0, 0, 7, 1, 0, 0, 0, 0, 3,
     ]

def checkSudoku(q):
  for i in range(9):
    for j in range(1, 10):
      if q[i * 9 : i * 9 + 9].count(j) > 1:
        return False
      if [q[k * 9 + i]
        for k in range(9)].count(j) > 1:
        return False
      if [q[i//3*27 + i%3*3 + k//3*9 + k%3]
        for k in range(9)].count(j) > 1:
        return False
  return True

def solveSudoku(q):
  if not checkSudoku(q): return None
  try:    tgt = q.index(0)
  except: return q
  for i in range(1, 10):
    q[tgt] = i
    if solveSudoku(q): return q
  q[tgt] = 0
  return None

ans = solveSudoku(q0)
for i in range(9):
  print(ans[i * 9:i * 9 + 9])
```

同じものを別言語 (Ruby) で

```ruby
q0 = [7, 0, 0, 0, 0, 5, 9, 0, 0,
      0, 2, 0, 0, 9, 0, 0, 3, 0,
      0, 0, 3, 2, 0, 6, 0, 0, 5,
      1, 3, 0, 0, 0, 0, 2, 0, 0,
      0, 0, 5, 0, 0, 0, 6, 0, 0,
      0, 0, 4, 0, 0, 0, 0, 7, 8,
      4, 0, 0, 5, 0, 2, 7, 0, 0,
      0, 9, 0, 0, 6, 0, 0, 4, 0,
      0, 0, 7, 1, 0, 0, 0, 0, 3,
     ]

def check_sudoku(ary)
 9.times do |i|
  a = ary[i * 9, 9]
  b = (0..8).map {|j| ary[j * 9 + i]}
  c = (0..8).map {|j|
   ary[i/3*27 + i%3*3 + j/3*9 + j%3]
  }
  (1..9).each do |j|
   return false if
    a.count(j) > 1 ||
    b.count(j) > 1 || c.count(j) > 1
  end
 end
 true
end

def solve_sudoku(q)
  return nil unless check_sudoku(q)
  return q unless tgt = q.index(0)
  ans = nil
  (1..9).each do |i|
    q[tgt] = i
    return ans if ans = solve_sudoku(q)
  end
  q[tgt] = 0
  nil
end

ans = solve_sudoku(q0)
9.times do |i|
  p ans[i * 9, 9]
end
```

プログラムとは

プログラム【program(me)】(広辞苑第六版)

1. 番組。予定。計画。
2. 目録。計画表。また、演劇・音楽会などの内容を解説した小冊子。
3. **コンピューターに対して、どのような手順で仕事をすべきかを、機械が解読できるような特別の言語などで指示するもの。**

機械が解読できるような特別の言語 = プログラミング言語

- プログラムを作成するために作られた**形式言語**。
- 用途や思想により様々なものがある。
- 最近では**ビジュアルプログラミング言語**も人気。

「プログラミング言語」ってどんなもの？

こんなものです。

人間向けの言葉:

1. **“Ready” と表示。**
2. **“Set” と表示。**
3. **“Go” と表示。**

子ども向けマイコンの
プログラミング言語:

Python:

「プログラミング言語」ってどんなもの？

こんなものです。

人間向けの言葉:

1. “Ready” と表示。
2. “Set” と表示。
3. “Go” と表示。

子ども向けマイコンの
プログラミング言語:

Python:

「プログラミング言語」ってどんなもの？

こんなものです。

人間向けの言葉:

1. “Ready” と表示。
2. “Set” と表示。
3. “Go” と表示。

子ども向けマイコンの
プログラミング言語:

Python:

```
print("Ready")
print("Set")
print("Go")
```

$0+1+\ldots+9$ in C

```c
#include <stdio.h>
int main()
{
  int sum;
  int i;
  for (i = 0, sum = 0; i < 10; i++) {
    sum = sum + i;
  }
  printf("%d\n", sum);
  return 0;
}
```

$0 + 1 + \ldots + 9$ in Rust

```
fn main() {
  let mut sum = 0;
  for i in 0..10 {
    sum = sum + i;
  }
  println!("{}", sum);
}
```

$0 + 1 + \ldots + 9$ in Java

```
public class Sum {
    public static void main(String[] args) {
      int i;
      int sum;
      for (i = 0, sum = 0; i < 10; i++) {
          sum = sum + i;
      }
      System.out.println(sum);
    }
}
```

$0 + 1 + \ldots + 9$ in JavaScript

```
let sum = 0
for (let i = 0; i < 10; i++) {
    sum = sum + i
}
console.log(sum)
```

$0+1+\ldots+9$ in Python

```
sum = 0
for i in range(10):
    sum = sum + i
print(sum)
```

$0 + 1 + \ldots + 9$ in Ruby

```
sum = 0
(0..9).each {|i|
  sum = sum + i
}
puts sum
```

$0+1+\ldots+9$ in Swift

```
var sum = 0
(1...9).forEach{(i) in
    sum = sum + i
}
print(sum)
```

$0 + 1 + \ldots + 9$ in Haskell

```
main = print $ foldl1 (+) [0..9]
```

$0+1+\ldots+9$ in TeX

```
\newcount\sum
\newcount\i
\sum = 0
\i = 0
\loop
  \advance \sum \i
  \advance \i 1
\ifnum \i < 10 \repeat
\message{\the\sum}
\end
```

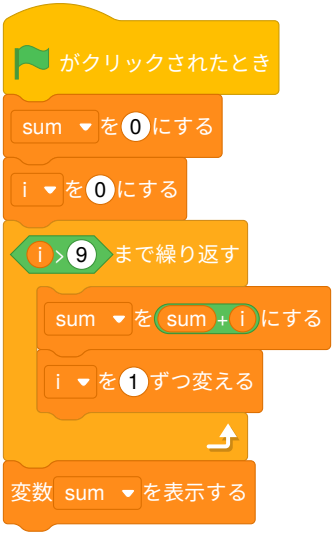
$0 + 1 + \ldots + 9$ in PostScript

```
0
0
10 {
    1 index
    add
    exch
    1 add
    exch
} repeat
=
```

$0+1+\ldots+9$ in (old) BASIC

```
10 S=0
20 FOR I=0 TO 9
30 S=S+I
40 NEXT I
50 PRINT S
```

$0 + 1 + \ldots + 9$ in Scratch



どこかでみたことあるような……？

https://www.overleaf.com/read/nqjwvwzxkzmy

どこかでみたことあるような……？

https://www.overleaf.com/read/nqjwvwzxkzmy

```
\begin{tikzpicture}
 \fontsize{80pt}{80pt}
 \clip(-2,-2)rectangle++(4,4);
 \node[fill=blue,minimum width=4cm,minimum height=4cm,inner sep=0pt,
       rounded corners=0mm](W){};
 \fontsize{60pt}{60pt}
\begin{scope}[rotate=5,every node/.append style={rotate=5}]
 \path(W.center)++(0.3,0.6)node[anchor=east,text=cyan,inner sep=2pt,
                        text opacity=0.9](PU){\sffamily\ebseries プ};
 \path(PU.east)node[anchor=west,text=cyan,inner sep=2pt,text opacity=0.9]
                                              (RO){\sffamily\ebseries 口};
 \path(PU.south)node[anchor=north,text=cyan,inner sep=2pt,text opacity=0.9]
                                              (EN){\sffamily\ebseries 演};
 \path(EN)-|(RO)node[midway,text=cyan,inner sep=2pt,text opacity=0.9]
                                               (I){\sffamily\bfseries I};
\end{scope}
 \fontsize{90pt}{90pt}
 \path(W.center)++(0,0.3)node[anchor=north,text=white,inner sep=0pt](CODE)
                              {\ttfamily\bfseries\textgreater\textunderscore};

 \fontsize{20pt}{20pt}
 \path(W.north west)++(0.77,-0.77)node[inner sep=3pt,minimum width=5cm,
                      rotate=45,text=yellow,fill=red]{\sffamily\bfseries 2024};
 \fontsize{10pt}{10pt}
 \path(W.north east)node[anchor=north east,text=white]
                                                    {\sffamily\ebseries 小林＆瀬尾};
\end{tikzpicture}
```

Python プログラミング

いきなり余談

以下はだいたい同じ意味。

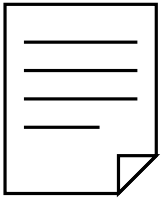
- **プログラム**を作る（組む）。
- **プログラミング**（を）する。

以下は誤用。

- **プログラミング**を作る（組む）。

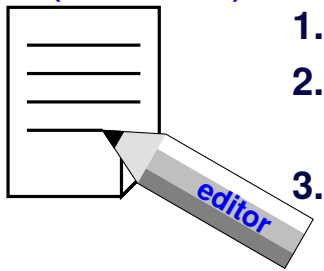
プログラムの作り方

program (source code)



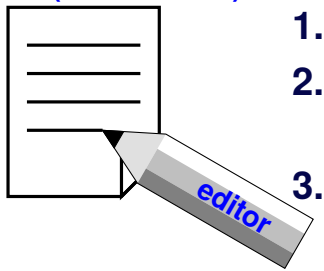
1. プログラムは**ファイル**として作る。
2. それを**ソース**（**ソースコード**や**コード**、あるいは**ソースファイル**など）という。
3. ソースを編集するには**エディタ**というソフト（アプリ）を使う。

プログラムの作り方



1. プログラムは**ファイル**として作る。
2. それを**ソース**（**ソースコード**や**コード**、あるいは**ソースファイル**など）という。
3. ソースを編集するには**エディタ**というソフト（アプリ）を使う。

プログラムの作り方



1. プログラムは**ファイル**として作る。
2. それを**ソース**（**ソースコード**や**コード**、あるいは**ソースファイル**など）という。
3. ソースを編集するには**エディタ**というソフト（アプリ）を使う。

この授業ではエディタとして**Visual Studio Code** (VSCode) というアプリを使う。

はじめての Python スクリプト (プログラム)

```
print("hello, world")
```

このプログラムを**作って**、**実行**しよう！

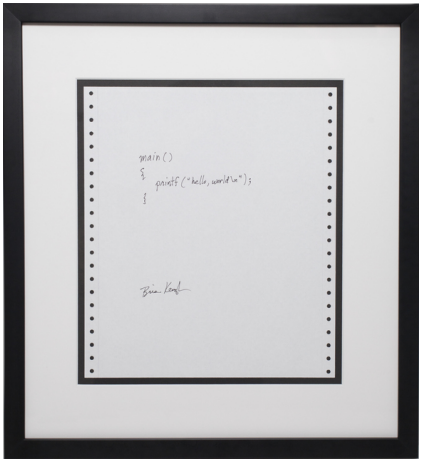
はじめての Python スクリプト (プログラム)

```
print("hello, world")
```

このプログラムを**作って**、**実行**しよう！

ただここからが少し長い。

コラム: hello, world



カーニハン自身による hello, world プログラム
img src=https://ja.wikipedia.org

プログラミングの入門書の最初に出てくる例題のお約束として **hello, world** と表示するものがあります。これは C 言語を開発した Kerninghan と Ritchie の歴史的名著「プログラミング言語 C」に出てくる最初の例題プログラム

```
#include <stdio.h>
main()
{
    printf("hello, world\n");
}
```

が由来とされています。いい加減な例ではなく、由緒正しいものなのです。

ターミナル（端末）を開く

この授業のすべての作業の司令塔となるアプリ

- **Windows** スタートボタン > Debian
- **macOS** Dock > Launchpad > その他 > ターミナル

- ターミナルからはいろいろな**コマンド**が入力できる。
- ターミナルを開いた直後は**ホームディレクトリ**というフォルダが作業場 (ワーキングディレクトリ) になっている。

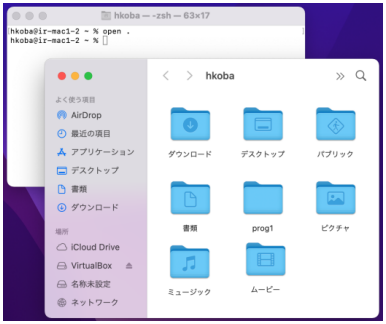
※多少私見が入りますが、**「フォルダ」と「ディレクトリ」は同義語**と思っていいです。ただ、前後につながる単語で慣習として「フォルダ」と「ディレクトリ」を使い分ける感じです。ホームディレクトリということは多いけれど、ホームフォルダということは（なくはないけれど）あまりない、というところ。

ホームディレクトリの確認

こうやると見えるのが**ホームディレクトリ**。この後の作業はすべてホームディレクトリの内側で行う。

ターミナルでの作業 (末尾のピリオド忘れに注意)

- Windows `explorer.exe .` Enter
  → 「.bash_history」「.bash_logout」など、いくつかのファイルが見える。
- macOS `open .` Enter
  →「ダウンロード」「デスクトップ」など、いくつかのフォルダが見える。



macOS のホームディレクトリの例

- Explorer や Finder でホームディレクトリを開くのはこのやり方が便利。
- Explorer や Finder でホームディレクトリを開く方法は他にもある。

ホームディレクトリに本授業ようフォルダを作る (初回のみの作業)

Explorer や Finder のウィンドウにも注目

ターミナルでの作業 1

pwd Enter

pwd=現在の作業フォルダ(ワーキングディレクトリ)フォルダを確認するコマンド

ターミナルでの作業 2

mkdir prog1 Enter

mkdir= フォルダを作るコマンド

ターミナルでの作業 3

ls Enter

ls=現在の作業フォルダ(ワーキングディレクトリ)の内容を表示するコマンド

作業フォルダを prog1 に切り替える (毎回やる作業)

ターミナルでの作業 1

こうすると、この後のターミナルでの操作は**prog1**フォルダ内で行われるようになる。

cd prog1 Enter

cd=作業フォルダ(ワーキングディレクトリ)フォルダを移る (切り替える) コマンド

おまけと復習: ここまでのコマンドの意味

- **pwd**:
- **mkdir**:
- **ls**:
- **cd**:

作業フォルダを prog1 に切り替える (毎回やる作業)

ターミナルでの作業 1

こうすると、この後のターミナルでの操作は**prog1**フォルダ内で行われるようになる。

cd prog1 Enter

cd=作業フォルダ(ワーキングディレクトリ)フォルダを移る (切り替える) コマンド

おまけと復習: ここまでのコマンドの意味

- **pwd**: Print Working Directory
- **mkdir**:
- **ls**:
- **cd**:

作業フォルダを prog1 に切り替える (毎回やる作業)

ターミナルでの作業 1

こうすると、この後のターミナルでの操作は**prog1**フォルダ内で行われるようになる。

cd prog1 Enter

cd=作業フォルダ(ワーキングディレクトリ)フォルダを移る (切り替える) コマンド

おまけと復習: ここまでのコマンドの意味

- **pwd**: Print Working Directory
- **mkdir**: MaKe DIRectory
- **ls**:
- **cd**:

作業フォルダを prog1 に切り替える (毎回やる作業)

ターミナルでの作業 1

こうすると、この後のターミナルでの操作は**prog1**フォルダ内で行われるようになる。

cd prog1 Enter

cd=作業フォルダ(ワーキングディレクトリ)フォルダを移る (切り替える) コマンド

おまけと復習: ここまでのコマンドの意味

- **pwd**: Print Working Directory
- **mkdir**: MaKe DIRectory
- **ls**: LiSt
- **cd**:

作業フォルダを prog1 に切り替える (毎回やる作業)

ターミナルでの作業 1

こうすると、この後のターミナルでの操作は**prog1**フォルダ内で行われるようになる。

cd prog1 Enter

cd=作業フォルダ(ワーキングディレクトリ)フォルダを移る (切り替える) コマンド

おまけと復習: ここまでのコマンドの意味

- **pwd**: Print Working Directory
- **mkdir**: MaKe DIRectory
- **ls**: LiSt
- **cd**: Change Directory

エディタを開き、作業フォルダを指定する。

いよいよプログラミングらしくなってきた！

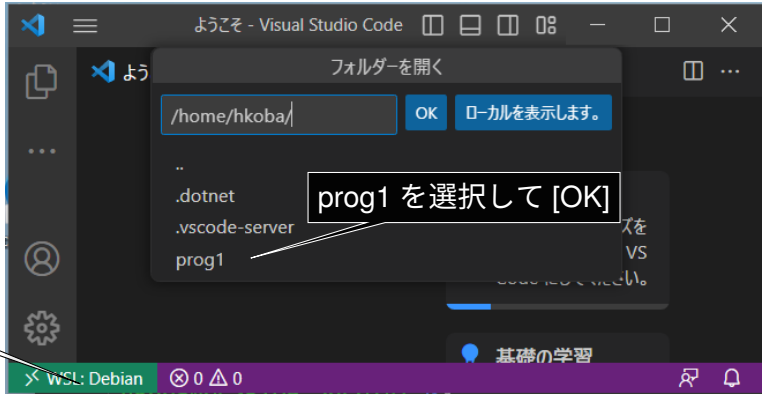
- VSCode を起動する。
- ファイル > フォルダーを開く として、**prog1 フォルダを開く**(下記参照)。
- 「フォルダー内の (中略) 信頼しますか?」と聞かれたら、信頼する。

VSCode の「フォルダーを開く」から「prog1 フォルダ」を開く方法.....

- **Windows** **次ページ参照**
- **macOS** (最初からホームディレクトリが見えているはず) > prog1

Windows の VSCode で prog1 フォルダを開く方法

- 右下の  **> WSL への接続 > フォルダーを開く**とすると下のようになる。



- prog1 を選択して [OK] で開く。(こうやって開いた VSCode のウィンドウを**WSL ウインドウ**という。)

Python 拡張機能をインストール

特に Windows の人は手順に注意

1. Windows の人は VSCode の**WSL ウインドウ**を開く。(macOS の人は VSCode の通常のウインドウを開く。)
2. (拡張機能) から “python” を検索し、Python 拡張機能を (Windows の人は WSL: Debian に) インストール する。

とりあえずここまでを振り返る。

煙に巻かれた視界を晴らそう。

今回だけやればいい環境構築

- ホームディレクトリでの**prog1**フォルダの作成
- Python 拡張機能のインストール

ここまでやったら初回の準備完了!

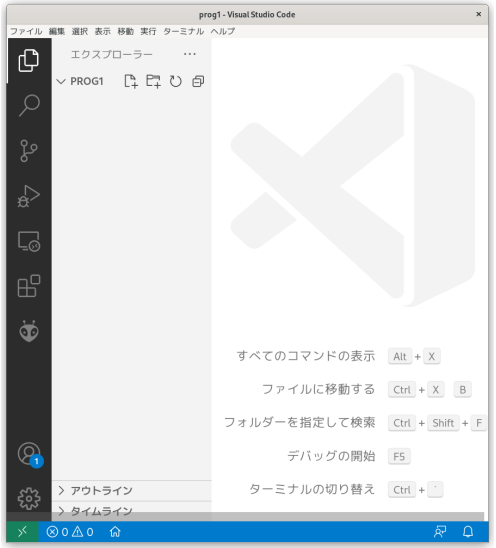
(予習復習含む) この授業でほぼ毎回やるであろうこと

- 端末を開いて、そのワーキングディレクトリを**prog1**にする。
- VSCode を開いて、その作業フォルダとして **prog1** を開く。(他で VSCode を使っていなければたぶん前回の続き状態で開くが、そうじゃないときは前のページのやり方で開く。)

ここまでやったら毎回の準備完了!

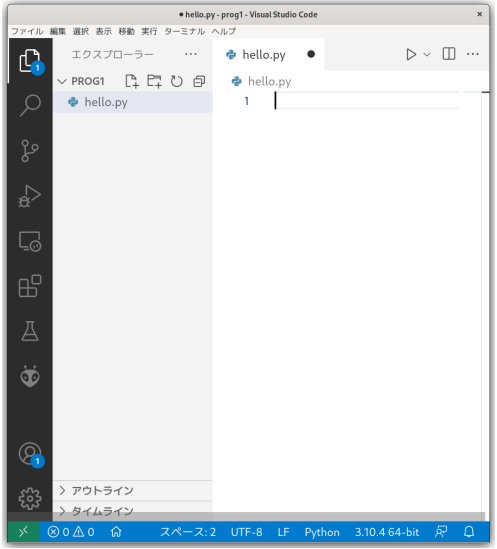
VSCode で新規ファイル（プログラムソースファイル）を作成する。

1. 右図のようになっているはず。
2. 縦に並んだアイコンの一番上が「VSCode のエクスプローラー」を開閉するボタン。→ 開いておく。
3. **PROG1** の横にある「新しいファイル」アイコンをクリック。
4. ファイル名を**hello.py**にする。このファイル名は作るプログラム毎に決める。(ファイル名は**半角アルファベット小文字と数字と特定の記号**の組み合わせで**末尾が.py**であること。)
5. **PROG1** の下の**hello.py**をクリックして **hello.py**を編集状態にする。



VSCode で新規ファイル（プログラムソースファイル）を作成する。

1. 右図のようになっているはず。
2. 縦に並んだアイコンの一番上が「VSCode のエクスプローラー」を開閉するボタン。→ 開いておく。
3. **PROG1** の横にある「新しいファイル」アイコンをクリック。
4. ファイル名を**hello.py**にする。このファイル名は作るプログラム毎に決める。(ファイル名は**半角アルファベット小文字と数字と特定の記号**の組み合わせで**末尾が.py**であること。)
5. **PROG1** の下の**hello.py**をクリックして **hello.py**を編集状態にする。



ようやく入力。hello, world

```
print("hello, world")
```

このプログラムを入力して、保存しよう！

そして実行！

ターミナルでの作業 1

`ls`	ソースファイルが作られたことを確認。

ターミナルでの作業 2

`python3 hello.py`	実行!

hello, world を詳しく見る。

```
print("hello, world")
```

[1]数学の関数とは意味が違うのであまり数学用語の関数という言葉は意識せず、まあそんなふうに呼ぶんだな、程度に受け取ってください。意味としては「命令」とか「コマンド」と同じようなものですが、Python ではそういう呼び方はしません。関数が正しい呼び方。

hello, world を詳しく見る。

```
print("hello, world")
```

- print は**何か**を表示する**関数**[1]。

[1]数学の関数とは意味が違うのであまり数学用語の関数という言葉は意識せず、まあそんなふうに呼ぶんだな、程度に受け取ってください。意味としては「命令」とか「コマンド」と同じようなものですが、Python ではそういう呼び方はしません。関数が正しい呼び方。

hello, world を詳しく見る。

```
print("hello, world")
```

- print は**何か**を表示する**関数**[1]。
- print(**何か**) のように丸括弧を使う。

[1]数学の関数とは意味が違うのであまり数学用語の関数という言葉は意識せず、まあそんなふうに呼ぶんだな、程度に受け取ってください。意味としては「命令」とか「コマンド」と同じようなものですが、Python ではそういう呼び方はしません。関数が正しい呼び方。

hello, world を詳しく見る。

```
print("hello, world")
```

- `print` は**何か**を表示する**関数**[1]。
- `print(`**何か**`)` のように丸括弧を使う。
- **何か** の部分として、「引用符 (`"`) で好きな言葉を囲んだもの (**文字列**)」が書ける。

[1] 数学の関数とは意味が違うのであまり数学用語の関数という言葉は意識せず、まあそんなふうに呼ぶんだな、程度に受け取ってください。意味としては「命令」とか「コマンド」と同じようなものですが、Python ではそういう呼び方はしません。関数が正しい呼び方。

hello, world を詳しく見る。

```
print("hello, world")
```

- print は**何か**を表示する**関数**[1]。
- print (**何か**) のように丸括弧を使う。
- **何か** の部分として、「引用符 (") で好きな言葉を囲んだもの (**文字列**)」が書ける。
- **何か** のことを **引数**(ひきすう) と言う。

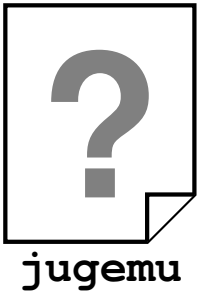
[1] 数学の関数とは意味が違うのであまり数学用語の関数という言葉は意識せず、まあそんなふうに呼ぶんだな、程度に受け取ってください。意味としては「命令」とか「コマンド」と同じようなものですが、Python ではそういう呼び方はしません。関数が正しい呼び方。

coffee break

拡張子

名は体を表すか？

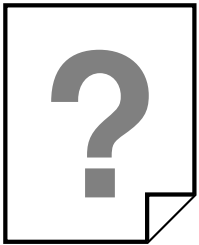
Q. このファイルに記録されているデータは何でしょう？



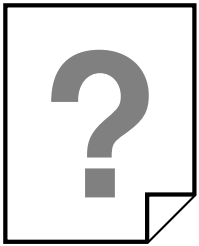
Q. このファイルに記録されているデータは何でしょう？

寿限無、寿限無、五劫の擦り切れ、海砂利水魚の…

1. テキストデータ



Q. このファイルに記録されているデータは何でしょう？

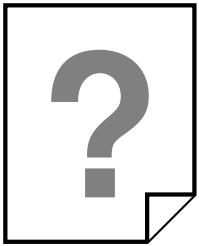


寿限無、寿限無、五劫の擦り切れ、海砂利水魚の…

1. テキストデータ

2. 動画データ

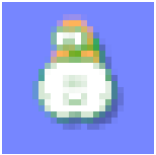
Q. このファイルに記録されているデータは何でしょう？



寿限無、寿限無、五劫の擦り切れ、海砂利水魚の…

1. テキストデータ

2. 動画データ



3. 画像データ

Q. このファイルに記録されているデータは何でしょう？

寿限無、寿限無、五劫の擦り切れ、海砂利水魚の…

1. テキストデータ

jugemu

2. 動画データ

3. 画像データ

ファイル名だけでは中身がわからない！

拡張子: ファイル名に「種別」情報を付加する慣習

OS によって扱いは異なるので注意しましょう。

拡張子のざっくりとした説明

- ファイル名の末尾に.py のように「ドット + 数文字」をつける。
- この部分を**拡張子**という。
- 人間や、コンピュータが、そのファイルの種類を識別するのに使われる。
- **Windows の初期状態では「非表示」になっている**(ので、表示させましょう)。

jugemu.txt

jugemu.mp4

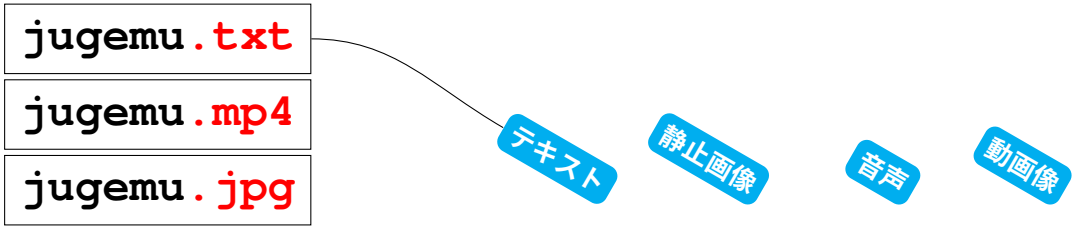
jugemu.jpg

拡張子: ファイル名に「種別」情報を付加する慣習

OS によって扱いは異なるので注意しましょう。

拡張子のざっくりとした説明

- ファイル名の末尾に.py のように「ドット + 数文字」をつける。
- この部分を**拡張子**という。
- 人間や、コンピュータが、そのファイルの種類を識別するのに使われる。
- **Windows の初期状態では「非表示」になっている**(ので、表示させましょう)。

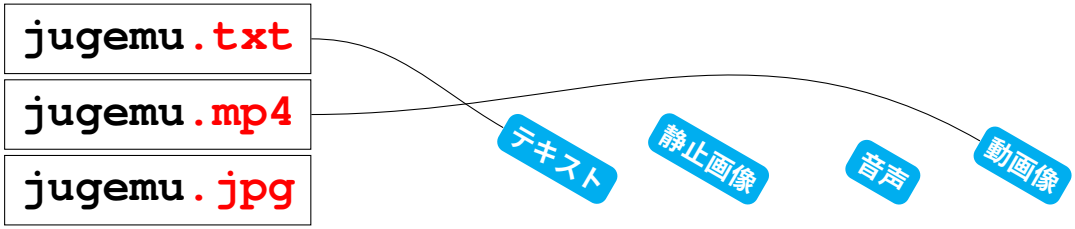


拡張子: ファイル名に「種別」情報を付加する慣習

OS によって扱いは異なるので注意しましょう。

拡張子のざっくりとした説明

- ファイル名の末尾に.py のように「ドット + 数文字」をつける。
- この部分を**拡張子**という。
- 人間や、コンピュータが、そのファイルの種類を識別するのに使われる。
- **Windows の初期状態では「非表示」になっている**(ので、表示させましょう)。

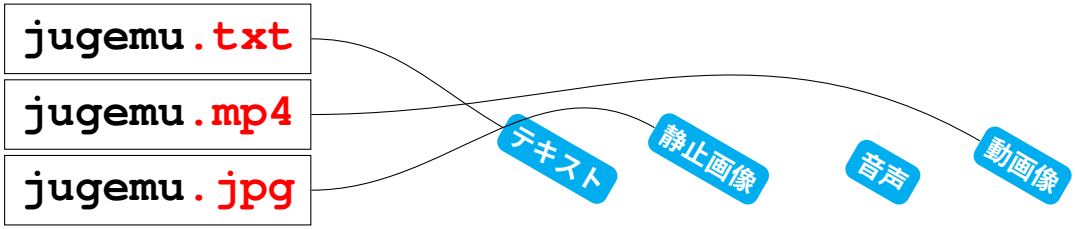


拡張子: ファイル名に「種別」情報を付加する慣習

OS によって扱いは異なるので注意しましょう。

拡張子のざっくりとした説明

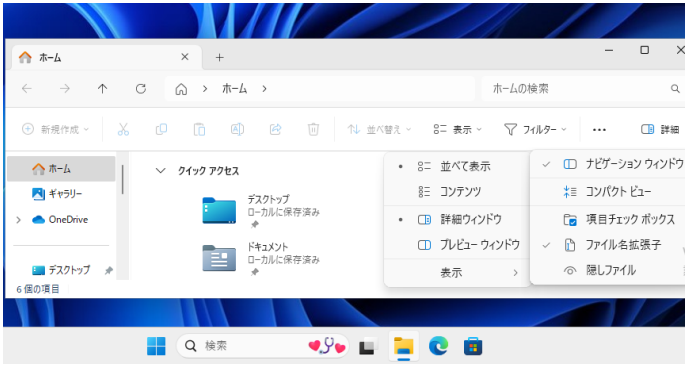
- ファイル名の末尾に.py のように「ドット + 数文字」をつける。
- この部分を**拡張子**という。
- 人間や、コンピュータが、そのファイルの種類を識別するのに使われる。
- **Windows の初期状態では「非表示」になっている**(ので、表示させましょう)。



Windows で拡張子を表示させる。

Windows の人は必ずやっておきましょう。

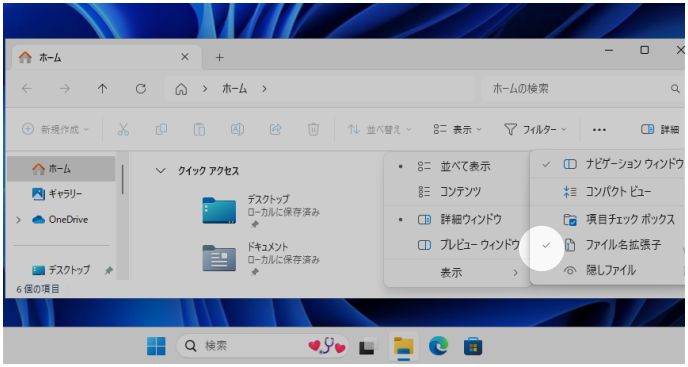
1. エクスプローラ（画面下に並んでいるアイコンのフォルダ型のもの）を開く。
2. 「表示」>「表示」メニューを開く。
3. 「ファイル名拡張子」にチェックを入れる。



Windows で拡張子を表示させる。

Windows の人は必ずやっておきましょう。

1. エクスプローラ（画面下に並んでいるアイコンのフォルダ型のもの）を開く。
2. 「表示」>「表示」メニューを開く。
3. 「ファイル名拡張子」にチェックを入れる。



補足と注意事項

- 拡張子はあくまでも**単に名前の一部**であり、データの中身を保証するものではありません。(これを使った攻撃手法もありました。)
- かと言って、拡張子を消したり変えたりしてしまうとプログラム（アプリ）から開けなくなったりするので注意。(うっかり何かしでかしてしまっても、単なる名前なので元に戻せば ok。)
- 知って得する拡張子ミニ hack:
  - `.xlsx` や `.docx` などを `.zip` にすると…?
  - `.ai` を `.pdf` にすると…?

  (他にもあると思うのでいろいろ遊んでみるのも良いでしょう。)

(coffee break おわり)

課題

- paiza ラーニング > 講座一覧 > 新・Python 入門編 の 25 あるレッスンの一番最初「入門編 1」を修了せよ。
- 認定証を確認する がクリックできる状態になっていればok.
- 工大の Google 認証で作成した (paiza の) アカウントでやること。
- 締め切りは日曜日の 24 時。