「アルゴリズムとプログラム」

- 本格プログラミングに挑戦 -

Python入門


```
def total(a):
    b = 0
    for i in a:
        b = b + i
    return(b)


x = [0, 1, 2, 3, 4, 5, 6, 7]

sum = total(x)
print(sum)
```

どうしてPython(1)

言語の標準機能を拡張する各種のライブラリーが用意されています。特にAI分野で豊富にそろっています。
Google,YouTubeやInstagramも、Pythonで開発されています。

tensorflow	ディープラーニングのフレームワーク
keras	ディープラーニングのフレームワーク
Jupyter	Webブラウザ常で操作できるPythonカーネル
Flask	Webアプリのフレームワーク
Django	Webアプリのフレームワーク
requests	Web(HTTP)ライブラリー
scikit-lean	機械学習ライブラリ
matplotlib	グラフライブラリー

ただし、スマホアプリはJava/JavaScriptが主流で、
Python用のライブラリーは今後に期待されます。

どうしてPython(2)

〇世界レベルのIEEEの2018年のプログラミング言語ランキングで1位です。
〇特にAI分野でPythonが使われるため、高収入プログラム言語になっています。
〇海外の大学・中学・高校での学習用プログラミング言語のトップです。他のプログラムより読みやすくて、動かしやすい。

国内では海外に比べてAI等が遅れているため、まだまだPythonのエンジニアが不足しています。

どうしてPython(3)

New

ブロック型ビジュアル言語
- 習得しやすい
- 見た目で理解しやすい
- すぐに楽しいものが作れる
- エラーが出にくい

テキスト言語
- 難しい
- プログラムの動作を考える必要がある
- いろいろな物が作れる
- プログラムの入力でもエラーがでる

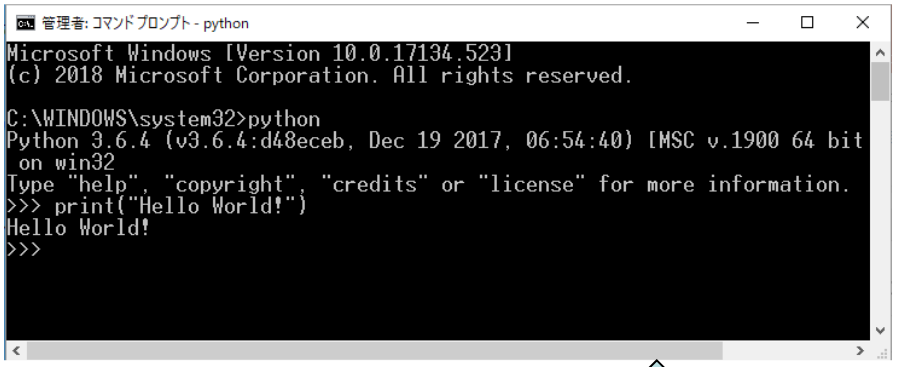
プログラミムの基本や考え方は同じ

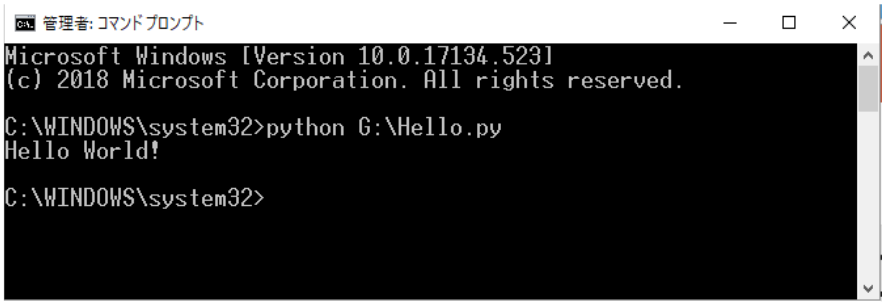
似ているので、一つの言語が使えると他のものを覚えるのも楽

Pythonを使ってみよう

方法1: Pythonシェルを使って、1行づつ入力して実行させる。

方法2: Pythonのプログラムファイルを一括して実行させる。





IDLE(Integraged Development Environment)
Pythonの統合開発環境
方法1も方法2も簡単に使えます

プログラム言語単体ではなく、各言語には統合開発環境が整備されていて、普通はそれを使って開発します。

学習の進め方

チェックシートが用意されています。チェックしながら学習してみましょう。

- 理解: 資料の内容を理解します。
- 確認・実行: 資料の内容をPCに入力して動作を確認します。
- 開発: 資料をみて、その課題プログラムを作成します。

python 入門□ (2018)□□自己チェックシート□□□□□□名前□□□□□□□□□□□□□

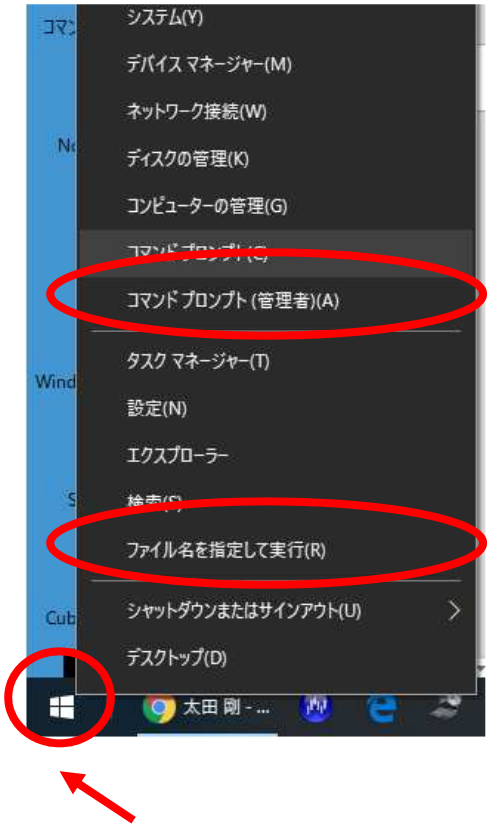
内容	チェック			備考*
	(理解)	(確認・実行*)	(開発)	
1. 変数と四則演算				
2. 文字列の定義				
3. キーボードからの入力				
4. プログラムファイルの作成と実行				
5. 文字/数字と数値				
6. プログラムの構造/フローチャートの復習				
7. Python の条件分岐				
8. チャレンジ: 正三角形の判断				
9. Python の繰り返し[for 文]				

確認・実行するところは、次の印がついています。

確認･実行

IDLE使ってみよう:起動

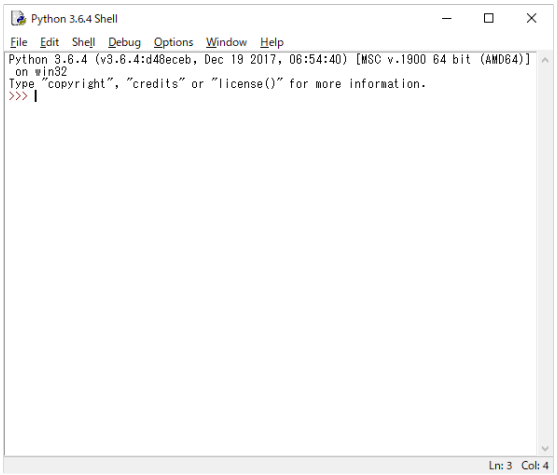
Windowsのコマンドプログラム(CGI:テキストベースの操作)を使うために、[コマンドプロンプト]又[ファイル名を指定して実行]を使って



ここにマウスカーソル持っていったから右クリック。

idle⏎

でIDLEを起動する



Pythonシェルの画面が起動する
(シェルウィンドウ)

IDLE使ってみよう:Pythonシェルの利用

シェルウィンドウでは一行づつプログラムを入力して、実行することができます。

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC
v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

プロンプト、ここに1行のプログラムを入れる。

初めてのPythonプログラム

確認・実行

```
>>> print("Hello World!")
Hello World!
>>> print(1+2)
3
>>>
```

これをプログラムとして入力すると青字の部分が実行結果

Print()　表示する

エラーが出た人はつぎのスライドを見て対応

エラーの対応(1)

シェルウィンドウでプログラムを1行づつ実行しているときは電卓みたいなモードです、エラーがあっても、次に正しく入力すれば問題ありません。

```
>>> pritn(1+2)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    pritn(1+2)
NameError: name 'pritn' is not defined
>>> print(1+2)
3
>>>
```

printの綴りが間違ってエラーになりますが、問題ありません。

正しく打ち直せばOKです。

エラーの対応(2)

Pythonはプログラムを1行づつ実行して、エラーがあると止まります。

```
>>> 1ban = 1
SyntaxError: invalid syntax
>>> x = 1
>>> print(X)
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    print(X)
NameError: name 'X' is not defined
>>> prnit(x)
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    prnit(x)
NameError: name 'prnit' is not defined
>>>
```

変数名が数字で始まっています。

変数はx(小文字)なのにprint(X)のX(大文字)なので違う変数として扱われて、変数が見つかりません。

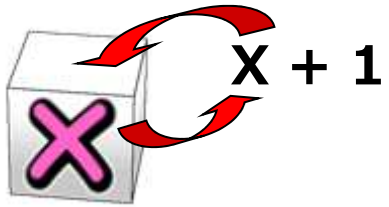
printの綴りが間違っています。

重要:
Pythonは大文字・小文字を区別します

1. 変数と四則演算: Scratchの復習

X = 1

Xと名前をつけた箱(変数)に1を入れる



X = X +1

(X ← X + 1のイメージ)

初めにXの箱(変数)の中を取り出し+1する。
計算結果をXの箱(変数)に入れなおす。

◎ScratchでのX = X +1

次の二つは同じ意味

変数への代入は普通の数学の=とは違う意味なのでイメージを示してみました。

1. 変数と四則演算(1): Python

プロンプトの後を入力して試してみよう

```
>>> x = 1
>>> print(x)
1
>>> x = x + 1
>>> print(x)
2
>>> x += 1
>>> print(x)
3
>>>
```

確認・実行

正しく動作しない、エラーがでる場合の対応は次のスライド

重要: 変数名の付け方
使える文字
・小文字英字　・大文字英字
・数字　・_(アンダーバー)
先頭に数字は使えない
○ abc ○Kakaku ○ total_a
× 1ban ×take@jp ×日本語

1. 変数と四則演算(2): Python

四則演算を確認しよう

確認・実行

```
>>> a = 6
>>> b = 2
>>> print(a + b)
8
>>> print(a - b)
4
>>> print(a * b)
12
>>> print(a / b)
3.0
>>>
```

足し算 +	+
引き算 -	-
掛け算 x	*
割り算 ÷	/

2. 文字列の定義

文字列の扱いを確認しよう

確認・実行

```
>>> print("Hello World!")
Hello World!
>>> a = "Hello"
>>> b = 'World!'
>>> c = a + b
>>> print(c)
HelloWorld!
>>> print("こんにちは")
こんにちは
>>>
```

・文字列は”(ダブルクォート)又は’(シングルクォート’で囲んで定義します。
・必ず同じ” 又は’でくくります。
・文字列同志をくっつける時は + を使います。
・文字列としては日本語も使用できます。

3. キーボードからの入力

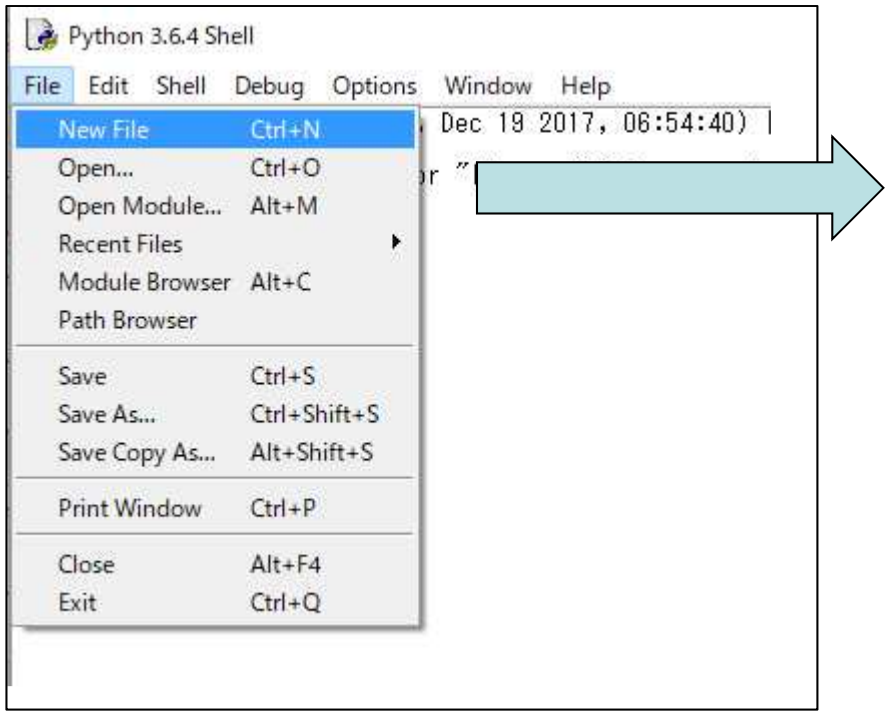
コーボードから入力してみよう

```
>>> a = input()
10
>>> print(a)
10
>>> b = input("Bを入力")
Bを入力20
>>> print(b)
20
>>> s = input("名前は")
名前はOta
>>> print(s)
Ota
>>>
```

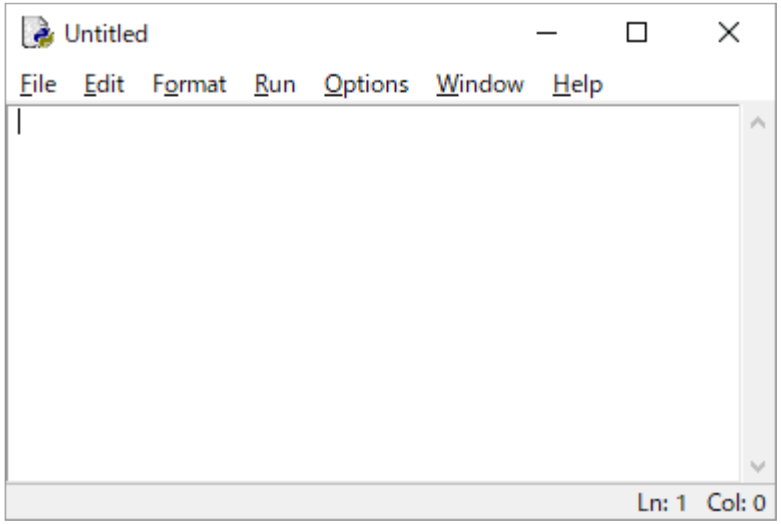
確認・実行

4. プログラムファイルの作成と実行(1)

プログラムファイルを作ってまとめて実行



Pythonシェル
(シェルウィンドウ)
New File: 新しいファイルの作成
Open: 既存のファイルを開く

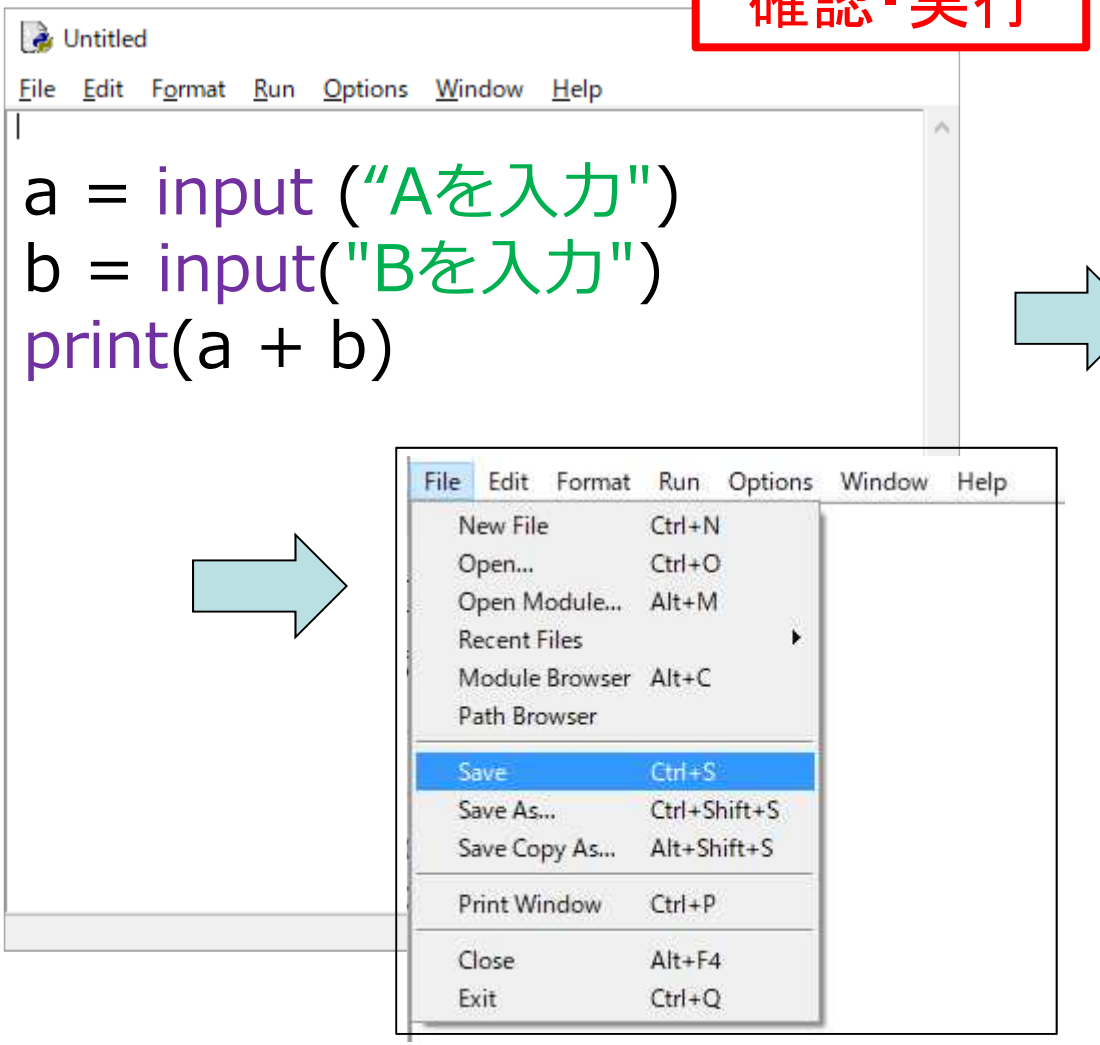


Python用簡易エディタ
(コードウィドウ)
プログラムを入力・変更する画面

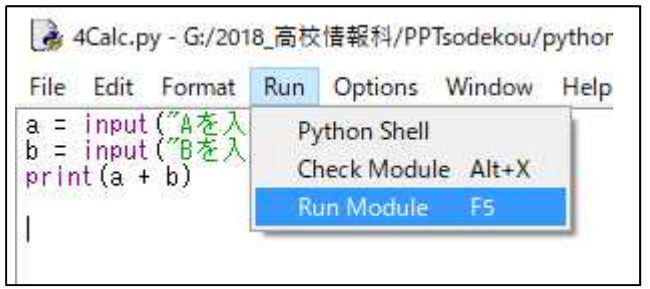
4. プログラムファイルの作成と実行

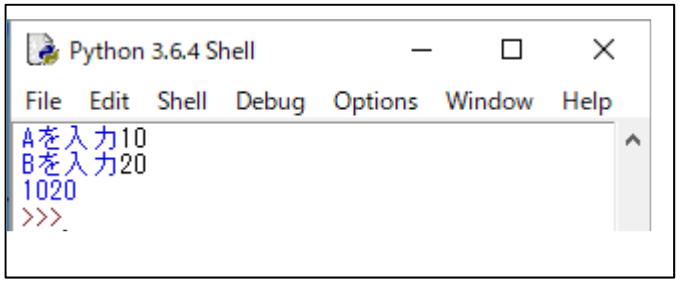
コードウィンドウにプログラムを入力して実行してみよう

確認・実行



実行しよう





シェルウィンドウで実行される(エラーがある場合も表示)

ファイルを保存しておこう(ドライブへ)

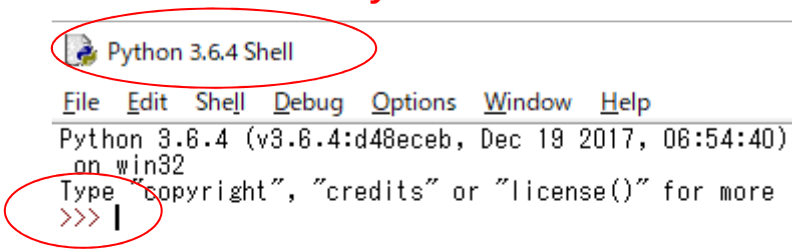
シェルウィンドとコードウィンドの見分け方

シェルウィンドとコードウィンドは見た目にていますが、違い違う使い方をします。しっかり区別しましょう。

シェルウィンドウ

・一行づつ入力して実行します。
・プログラムの実行結果(入力・出力)を出します。
・プログラムのエラーを表示します

ウィンドウの名前は
Python xxxx Shell

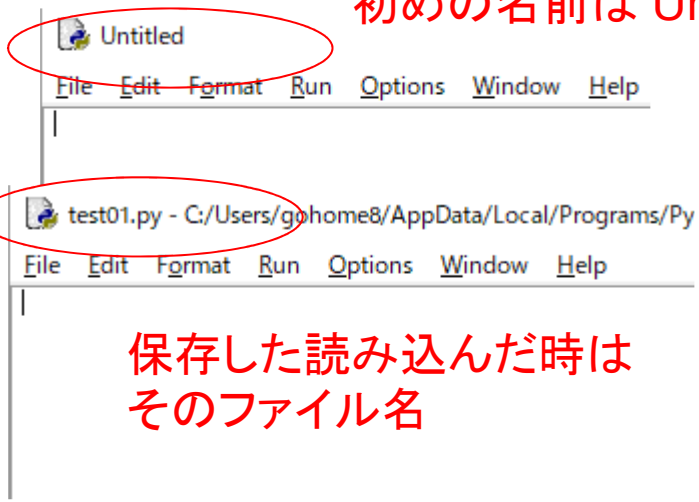


>>> のプロンプトが表示

コードウィンドウ

・まとまったプログラムの作成と編集をします。
・プログラムの保存と読み込みをします。
・プログラムの実行を指示します。
(実行結果はシェルウィンドウ)

初めの名前は Untieled

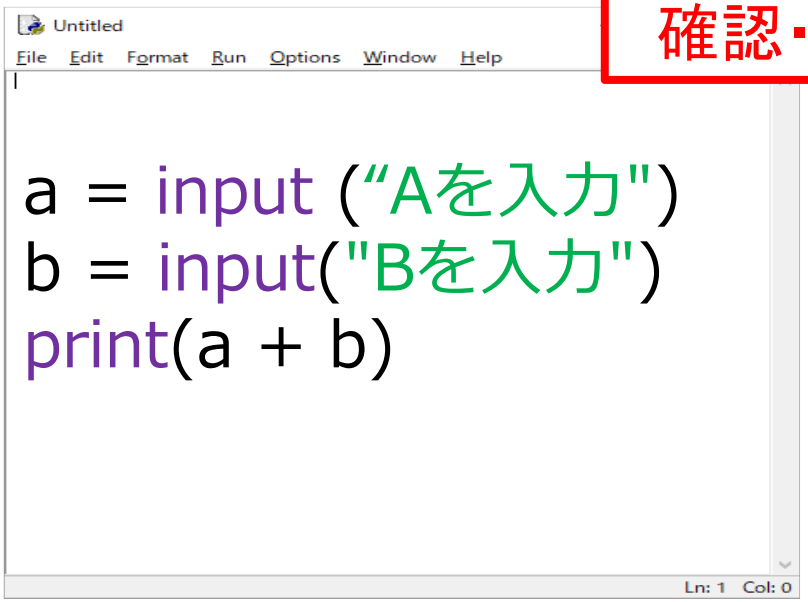


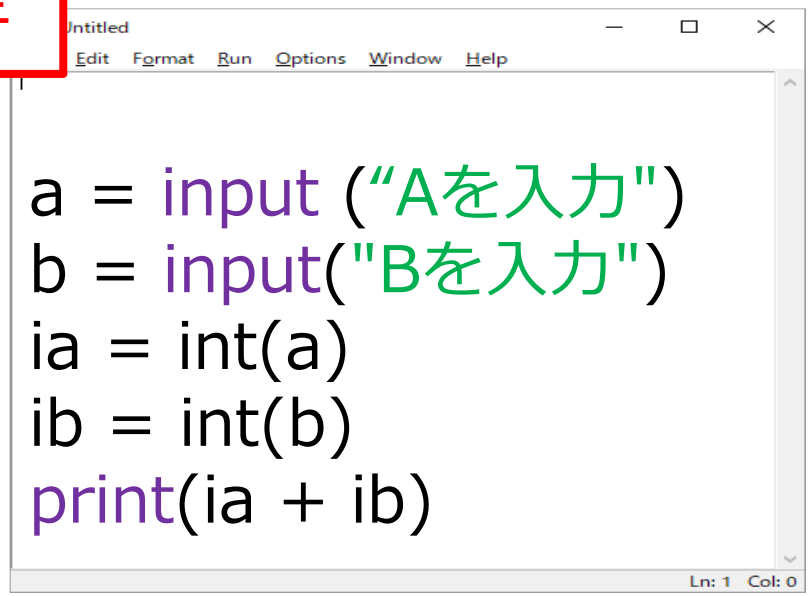
保存した読み込んだ時は
そのファイル名

5. 文字/数字と数値

前のプログラムの実行結果は少しおかしいと思いませんか?
次の二つのプログラムを作って比べてみよう

確認・実行

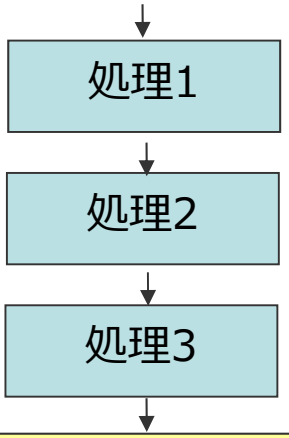




重要: inputはすべて文字列として入力する。
入力した数字を、計算で使えるような数値に変換するには
整数の場合: int()
小数の場合: float()
で変換する

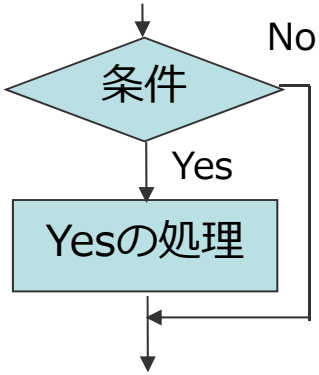
6. プログラムの構造/フローチャートの復習(1)

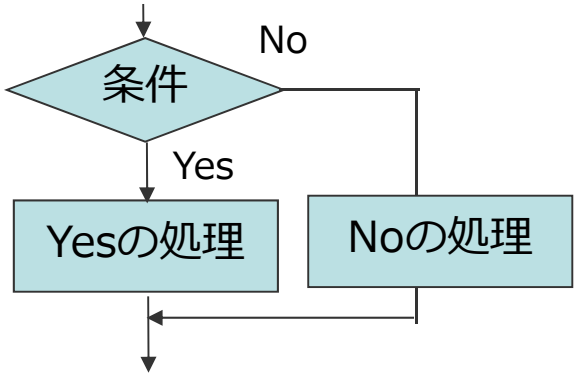
逐次構造(直線型)



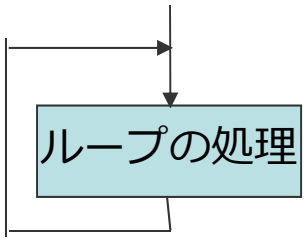
プログラムや人間の判断などのアルゴリズムは基本的に、逐次、選択:分岐、繰り返し(ループ)の組み合わせで表現できます。

選択構造:分岐

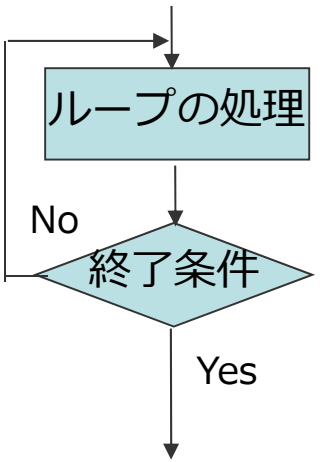




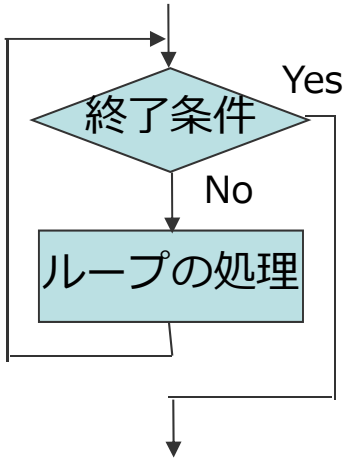
繰り返し構造(ループ)



無限繰り返し型



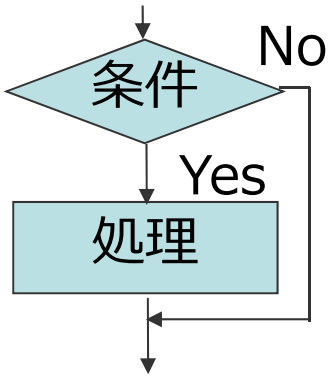
後判定型

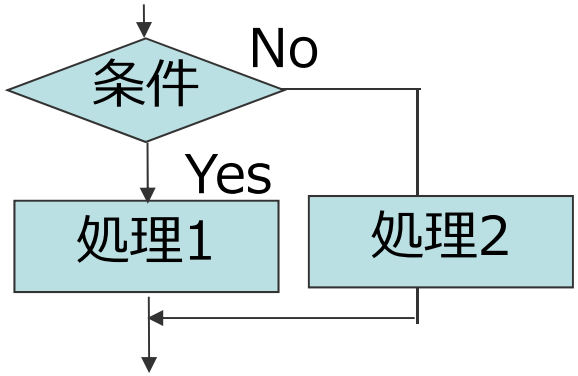


前判定型

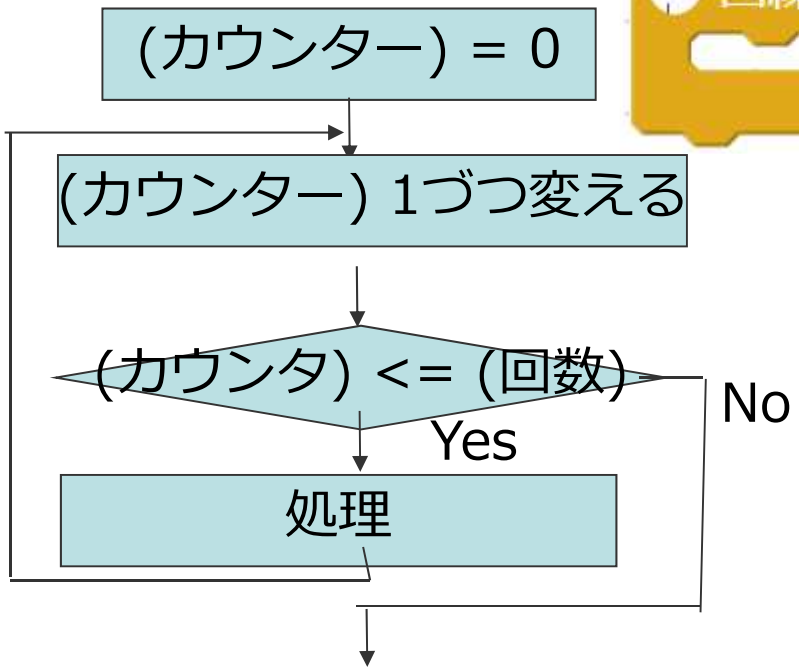
6. プログラムの構造/フローチャートの復習(2)

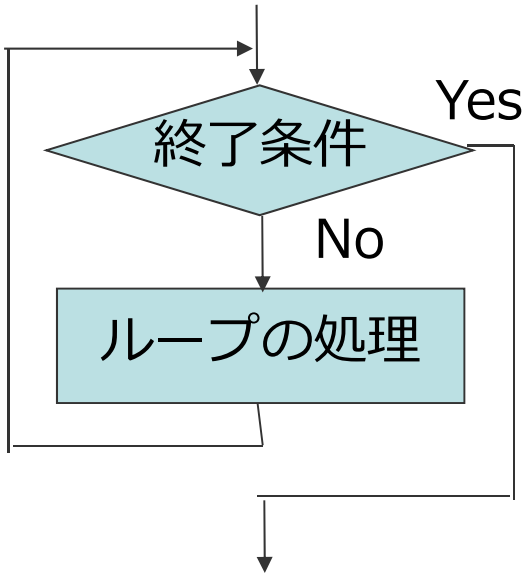
分岐





繰り返し:回数指定

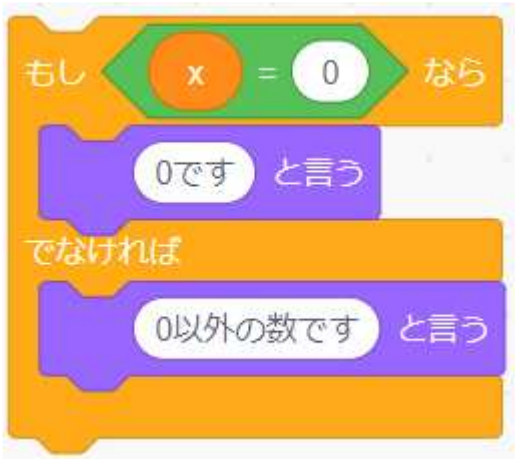




7. Pythonの条件分岐(1)

ScratchとPythonの条件分岐を比べてみました

Scratch



Python

```
if x > 0:
    print("正の数")
```

```
if x == 0:
    print("0です")
else:
    print("0以外の数です")
```

重要:
- ○ ifやelseの行の最後は: (コロン)をつけて明確にする
- ○ ifやelseの中で実行する命令は、4桁の字下げをする
  (pythonの絶対的なルール)

7. Pythonの条件分岐(2)

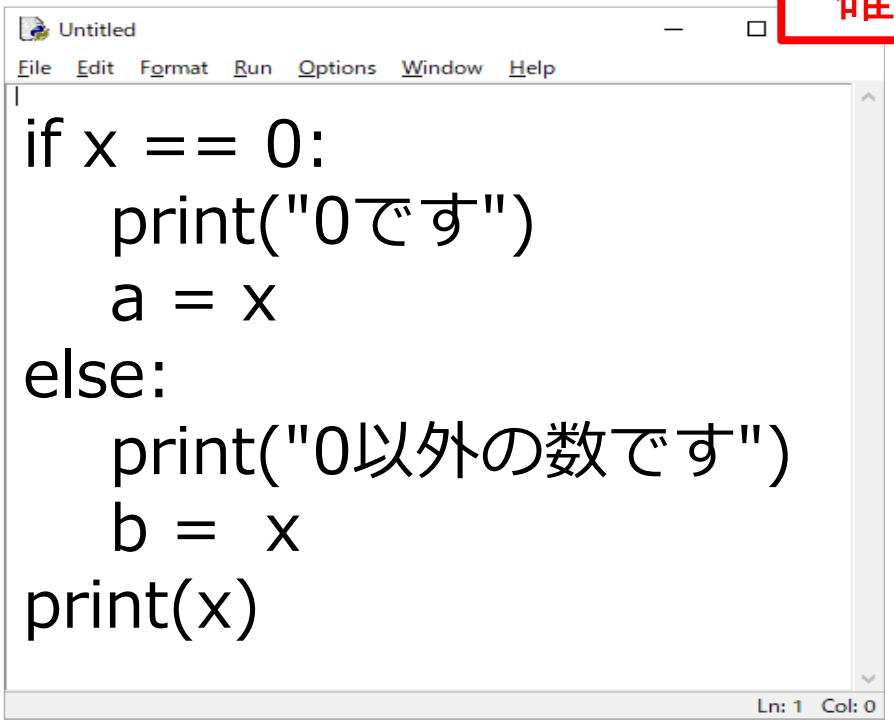
条件分岐で使う条件式を比べてみました

Scratch	Python
	x == 0
	x > 0
	x < 50
	a == 0 and b == 0
	a == 0 or b == 0
	not x == 0
	x != 0

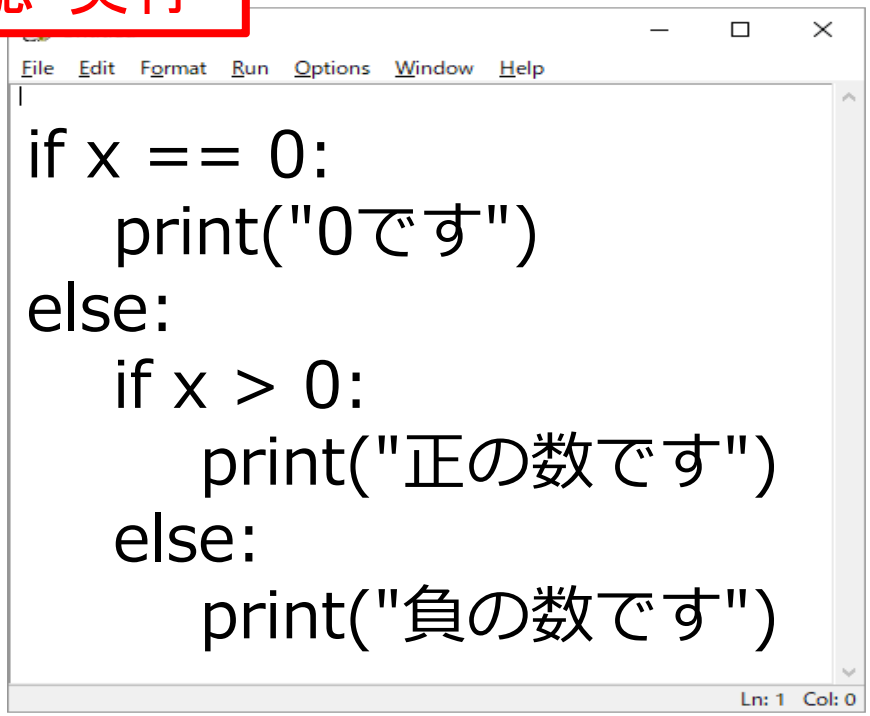
7. Pythonの条件分岐(3)

if文のいろいろな書き方

確認・実行



重要:
ifやelseの中で複数の命令を実行 =
4桁の字下げしたままにする。
(字下げをやめるとifやelseの中ではない)



重要:
ifやelseの中にif文を入れる =
4桁の字下げしたままにする。

8.　チャレンジ: 正三角形の判断(1)

a, b, cの3個の三辺の値を入力して、すべての値が同じ場合に、「正三角形」、そうでない場合は「正三角形じゃない」と表示するプログラムを作ってみよう。

フローチャートは作っても、作らなくてもいいです。

・a,b,cは1以上の整数を入力する前提で作っていいです。

8.　チャレンジ: 正三角形の判断(2)

Scratchで作った場合を参考にしてください。

次のスライドに
ヒントがあります。

ファイル提出

提出ファイル名
xxC08.py（xxは出席番号）

8. チャレンジ: 正三角形の判断(3):ヒント


```
ta = input (“Aの値は")
a = int(ta)
```

```
if a == b:
```

9. Pythonの繰り返し[for文](1)

ScratchとPythonの繰り返し(似たような動作)を比べてみました。 Pythonのプログラムを実行して確認してみよう

Scratch　　確認･実行　　Python


```
for x in range(10):
    print(x)
```

重要:range(10)
0から10未満の+1づつの整数を生成する


```
for x in range(1,11):
    print(x)
```

重要:range(1,11)
1から11未満の+1づつの整数を生成する

9. Pythonの繰り返し[for文](2)

range()をもっと詳しく見てみます。

一般形　range(開始値, 終了値, 増分)
　　開始値: 省略時 0, 増分: 省略時 1

例 range(5) :開始値:省略 0, 増分: 省略 1
　　0, 1, 2, 3, 4

例 range(1,5) :増分: 省略 1
　　1, 2, 3, 4

例 range(1, 5, 2)
　　1, 3

10.　チャレンジ: 1 から 10までの合計

チャレンジa
　1 から 10までの合計をfor文を使って計算する
プログラムを作ってください。

チャレンジb
　2から 10までの偶数の合計をfor文を使って計算する
プログラムを作ってください。

チャレンジbのみファイル提出

提出ファイル名
xxC10.py（xxは出席番号）

11. リスト(配列)(1)

PythonもScratchと同じようにリストが使えます

Scratch

Python

```
D = [ 1, "ABC", 100, "Hello"]
```

0	1
1	ABC
2	100
3	Hello

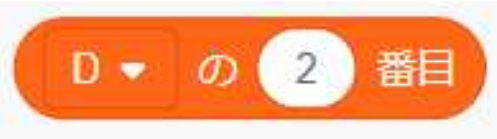
重要:
Pythonの場合、リスト内の箱の番号は0から始まります。


```
D = [ 1, 2, 3, 4]
D = list(range(1,5))
```

重要:
list()を使って、range()で生成した数列をリストの中にいれることができます。

11. リスト(配列)(2)

Scratchと同じようにリスト内の個々の要素が使えます

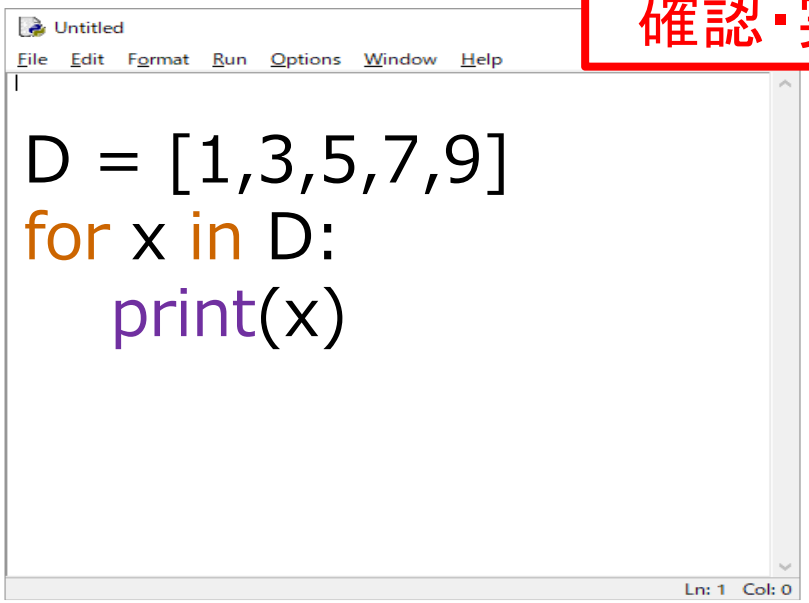
Scratch	Python
	D[1]
	D[1] = 100
	D.append(100)

重要:
Pythonの場合、リスト内の箱の番号は0から始まります。

12. コードウィンドウとシェルウィンドウ

コードウィンドウにプログラムを入力して実行してみよう。
その後、シェルウィンドウで状態を確認しよう。

コードウィンドウ



確認・実行

シェルウィンドウ

```
>>> print(D)
[1, 3, 5, 7, 9]
>>> print(D[2])
5
>>> D.append(100)
>>> print(D)
[1, 3, 5, 7, 9, 100]
>>>
```

重要:
for文は inの後の要素を一個づつ取り出して実行するのが本当の意味です。

重要:
プログラムが実行された状態が残り、シェルウィンドウで会話型でその状況を確認できます。

13. チャレンジ: リストの中の数の合計

10, 20, 30, 40, 50, 100が入っているリストを定義して、
その合計を求めるプログラムを作成してください。

ファイル提出

提出ファイル名
xxC13.py（xxは出席番号）

次のスライドに
ヒントがあります。

13. チャレンジ: リストの中の数の合計(ヒント)

10, 20, 30, 40, 50, 100が入っているリストを定義して、
その合計を求めるプログラムを作成してください。

数が入っているリストを作成する
kazu = [10, 20, 30, 40, 50, 100]

リストの中の値をすべて順番に取り出して処理する
for i in kazu:

14. 関数型言語とオブジェクト指向型言語(1)

いろいろなプログラミング言語がありますが、現在関数型やオブジェクト指向型言語が主流になっています。ScratchとPythonもこの二つの性質を持っています。Scratchでの関数とオブジェクトは何か見てみましょう。

関数

自分で作るブロックが関数のイメージです。ただし、戻り値がないので、厳密な意味で関数ではありません。

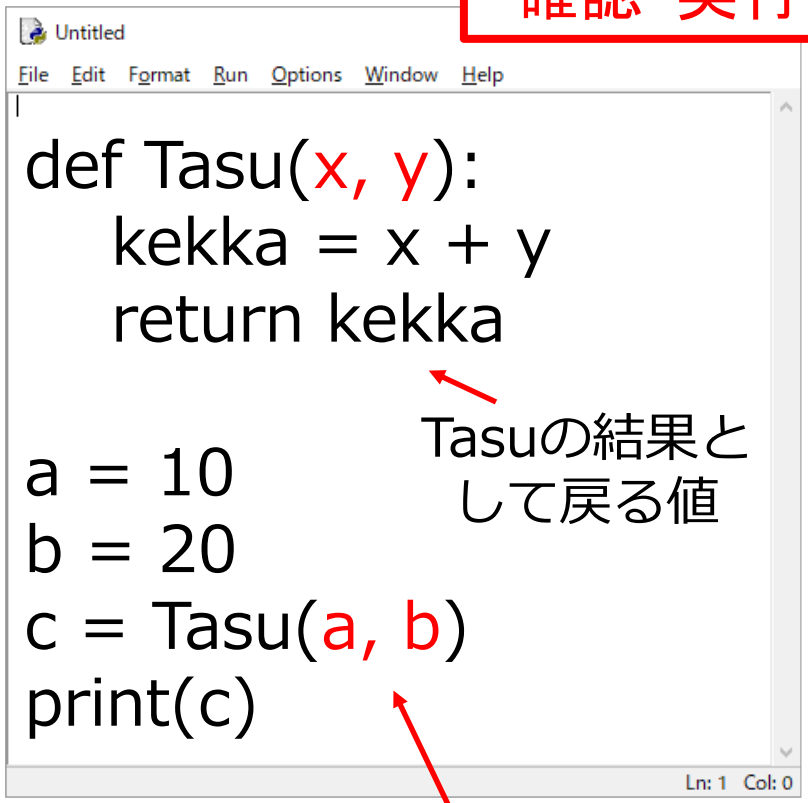
オブジェクト

コスチュームやスクリプトが集まった一つのスプライトがオブジェクトに対応します。この独立したオブジェクトが集まってプログラムとして動作します。

14. 関数型言語とオブジェクト指向型言語(2)

Pythonでも関数とオブジェクトが使用できます。ここでは関数だけを作っていきましょう。関数はExcelの=SUM()や=Average()を自分で作る感じです。

確認・実行



Tasuの結果として戻る値

x と yの値として引き渡される

関数定義方法

```
def 関数名(引数1, 引数2...):
    処理1
    処理2
    return 戻り値
```

(必ずしも引数と戻り値を持つ必要はありません)

重要:
defの中で複数の命令を実行 = 4桁の字下げしたままにする。
(字下げをやめるとdefの中ではない)

重要:
関数は呼び出す(使う)前に定義しておく必要があります

14. 関数型言語とオブジェクト指向型言語(3)

引数としてnを渡して、1からnの合計を計算する関数を定義して、使用するプログラムを作成しましょう。

ファイル提出

提出ファイル名
xxC14.py（xxは出席番号）

次のスライドに
ヒントがあります。

14. 関数型言語とオブジェクト指向型言語(4) ヒント

関数を使わない場合のプログラム例

```
tn = input("nを入力")
n = int(tn)
sum = 0
for i in range(1, n + 1):
    sum = sum +i
print(sum)
```

この部分を関数化します

15. Pythonの繰り返し[While文](1)

ScratchとPythonの繰り返し(似たような動作)を比べてみました。 Pythonのプログラムを実行して確認してみよう

Scratch

終了の時の条件を指定

Python

確認・実行

```
tn = input("nを入力")
n = int(tn)
while n > 0:
    print(n)
    n = n -1
```

繰り返しをする時の
条件を指定
(条件に合わないと終了)

15. Pythonの繰り返し[While文](2)

Pythonのプログラムを実行して、何をしているか
確認してみよう

確認･実行

```
sum = 0
i = 1
while i != 0:
    ti = input("iを入力")
    i = int(ti)
    sum = sum + i
print(sum)
```

プログラムが無限ループに入って止まらない場合は Ctrl + Cで強制終了できます。

より進んだ課題

課題1: 素数を求めるプログラム(割り算方式)

課題1素数を求めるプログラム(割り算方式)

Scratchのプログラムの場合と考え方は同じですが、Pythonの場合はもっとシンプルになります。たぶん11行ぐらい

次のスライドに
ヒントがあります。

ファイル提出

提出ファイル名
xxK01.py（xxは出席番号）

課題1素数を求めるプログラム(割り算方式)ヒント

プログラムを作る時の参考にしてください。


```
sosu = []　‘ 空のリストを作る

for i in range(2, n+1):
‘ rangeで2からnまで数を生成して繰り返す

for j in sosu:

‘リストの中身をjに取り出して繰り返す

i % j == 0　‘余りは%

sosu.append(i)
‘リストに追加する場合
```