



Python入門(1) Ver 1.0 (2019/2/24)

テキスト言語としてPythonを使ってみましょう。AI分野などで注目されているプログラミング言語です。
(Scratchでプログラミングできる人向けです)

- どうしてPython
- Pythonを使ってみよう
- IDLEを使ってみよう
- Pythonを学習してみよう

付録: Python3.0のインストール

```
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ブラウザ上で操作できる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-learn	機械学習ライブラリ
matplotlib	グラフィックライブラリー

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期待されます。

どうしてPython(2)

○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。

○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。

○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、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



プログラムの
基本や考え
方は同じ



テキスト言語

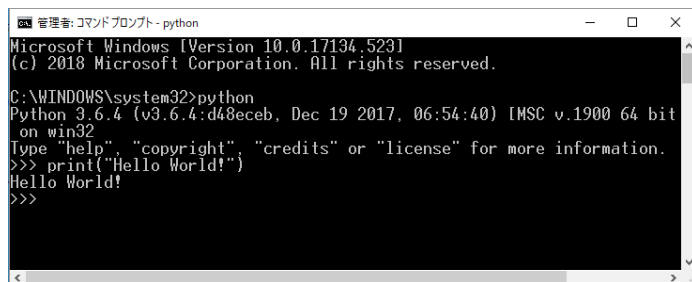
- 難しい
- プログラムの動作を考える必要がある
- いろいろな物が作れる
- プログラムの入力でもエラーがでる



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

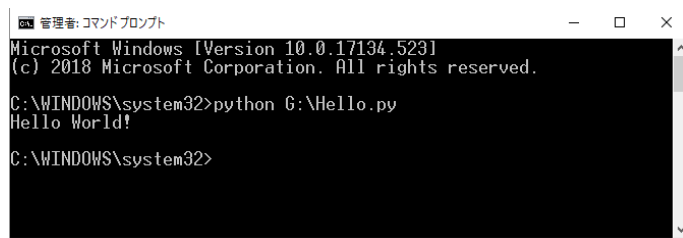
方法1: Pythonシェルを使って、1行ずつ入力して実行させる。



```
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSVC v.1900 64 bit
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World!")
Hello World!
>>>
```

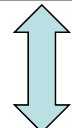
方法2: Pythonのプログラムファイルを一括して実行させる。



```
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>python G:\Hello.py
Hello World!

C:\WINDOWS\system32>
```



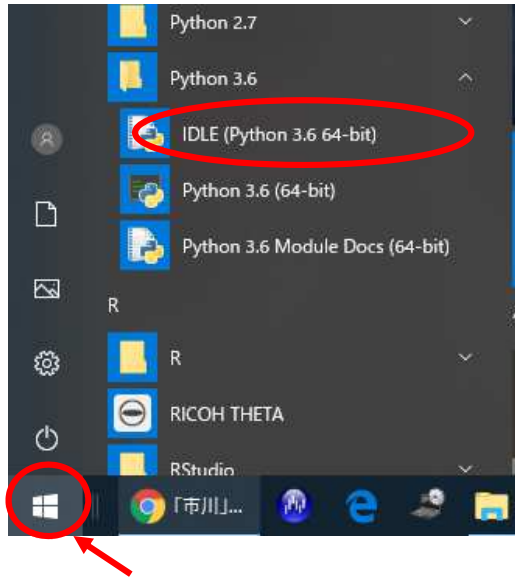
IDLE(Integrated Development Environment)

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。

IDLE使ってみよう:起動(1)



IDLEというPython
の開発ツールを起
動する

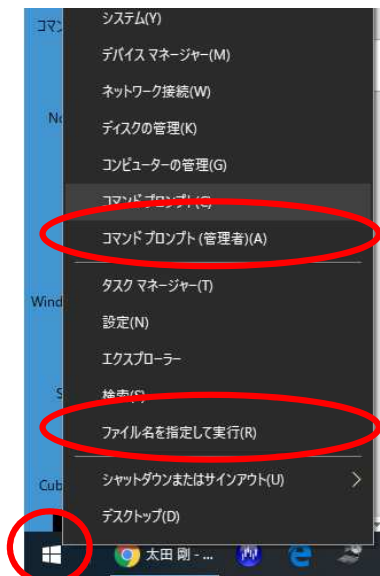
これがメニューが
なかったら、次の
ページ

ここにマウスカーソル持って
いったからクリック
(左クリック)。

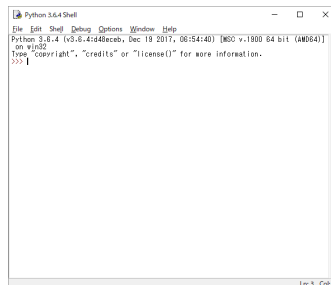
IDLE使ってみよう:起動(2)

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

idle ↩
でIDILを起動する



ここにマウスカーソル持っていったから右クリック。



Pythonシェルの画面が起動する
(シェルウィンドウ)

動かなかったらPythonがインストールされていないので、付録のインストールへ

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ず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行ず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。

```
>>> pritrn(1+2)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    pritrn(1+2)
NameError: name 'pritrn' 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行ずつ実行して、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

Pythonを学習してみよう

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。(次のページにあります)

- ・ **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	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2. 文字列の定義	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3. キーボードからの入力	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4. プログラムファイルの作成と実行	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5. 文字/数字と数値	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6. プログラムの構造/フローチャートの復習	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7. Python の条件分岐	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8. チャレンジ: 正三角形の判断	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9. Python の繰り返し[for文]	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

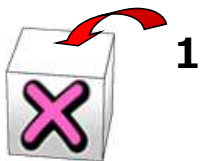
確認・実行するところは、次の印がついています。

確認・実行

チェックシート

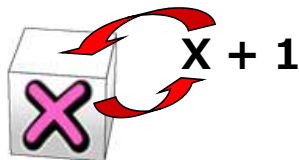
内容	チェック		
	理解	確認実行	開発
1. 変数と四則演算			
2. 文字列の定義			
3. キーボードからの入力			
4. プログラムファイルの作成と実行			
5. 文字/数字と数値			
6. プログラムの構造/フローチャートの復習			
7. Pythonの条件分岐			
8. チャレンジ: 正三角形の判断			
9. Pythonの繰り返し[for文]			
10. チャレンジ: 単純な1から10の合計			
11. 配列/リスト			
12. コードウィンドウとシェルウィンドウ			
13. チャレンジ: リストの中の数の合計			
14. 関数型言語とオブジェクト指向型言語			
15: Pythonの繰り返し[while文]			
16: チャレンジ: 素数を求めるプログラム			

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



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

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

1. 変数と四則演算(2): Python

四則演算を確認しよう

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

確認・実行

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 キーボードから入力してみよう

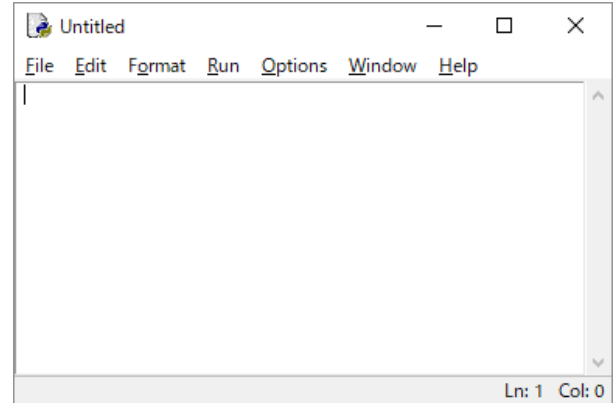
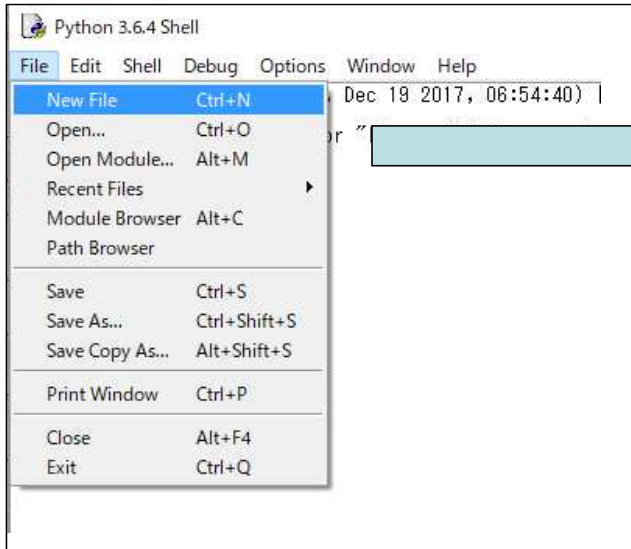
```
>>> a = input()
10
>>> print(a)
10
>>> b = input("Bを入力")
Bを入力20
>>> print(b)
20
>>> s = input("名前は")
名前はOta
>>> print(s)
Ota
>>>
```

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

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

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

```
File Edit Format Run Options Window Help
|
a = input ("Aを入力")
b = input("Bを入力")
print(a + b)
```

```
File Edit Format Run Options Window Help
a = input ("Aを入力")
b = input ("Bを入力")
print(a + b)
Python Shell
Check Module Alt+X
Run Module F5
```

```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Aを入力10
Bを入力20
1020
>>>
```

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)

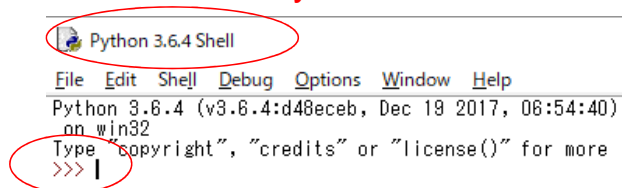
シェルウィンドとコードウィンドの見分け方

シェルウィンドとコードウィンドは見た目には見えますが、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

- ・一行ず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



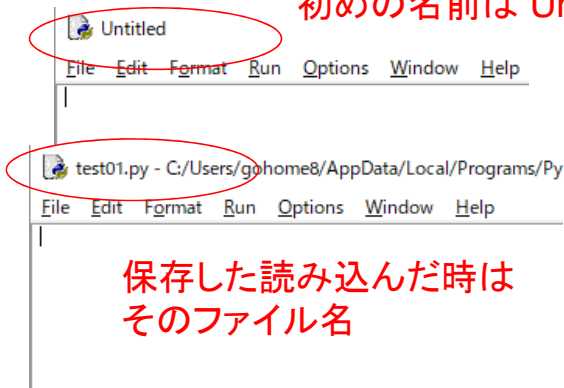
>>> のプロンプトが表示

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 Untitled



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

```
Untitled
File Edit Format Run Options Window Help

a = input ("Aを入力")
b = input("Bを入力")
print(a + b)

Ln: 1 Col: 0
```

```
Untitled
File Edit Format Run Options Window Help

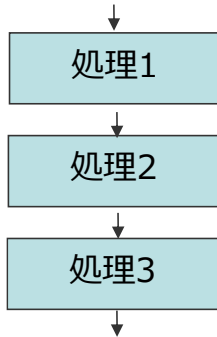
a = input ("Aを入力")
b = input("Bを入力")
ia = int(a)
ib = int(b)
print(ia + ib)

Ln: 1 Col: 0
```

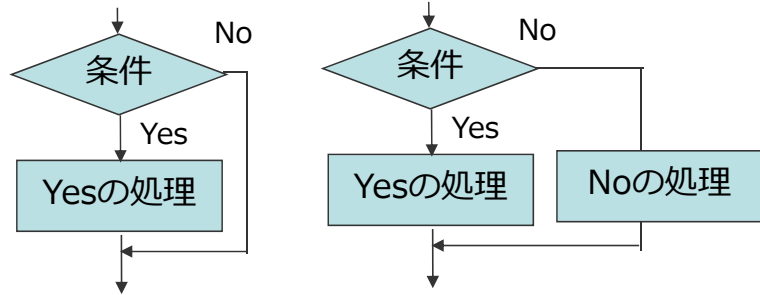
重要: inputはすべて文字列として入力する。
入力した数字を、計算で使えるような数値に変換するには
整数の場合: int()
小数の場合: float()
で変換する

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)



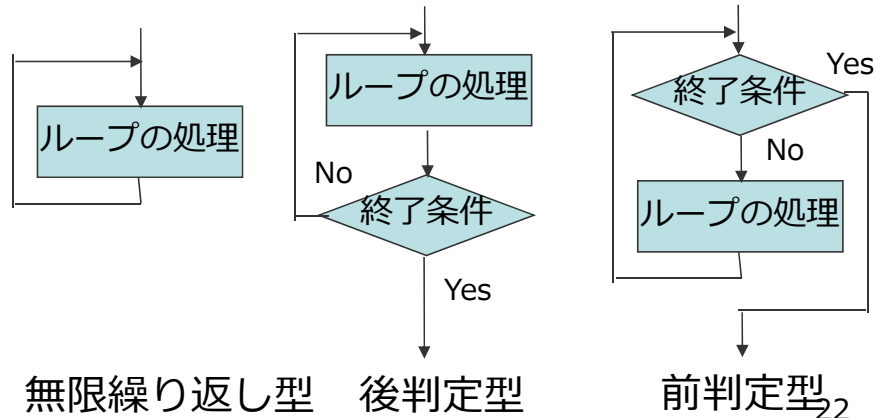
選択構造:分岐



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。

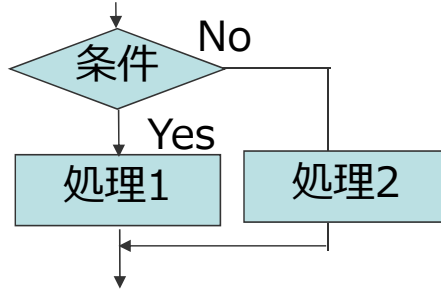
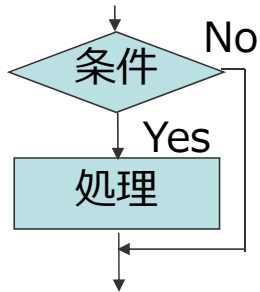


繰り返し構造(ループ)

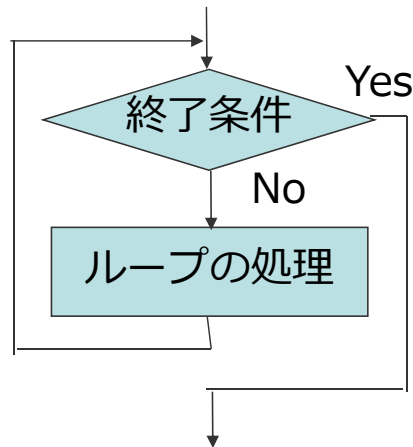
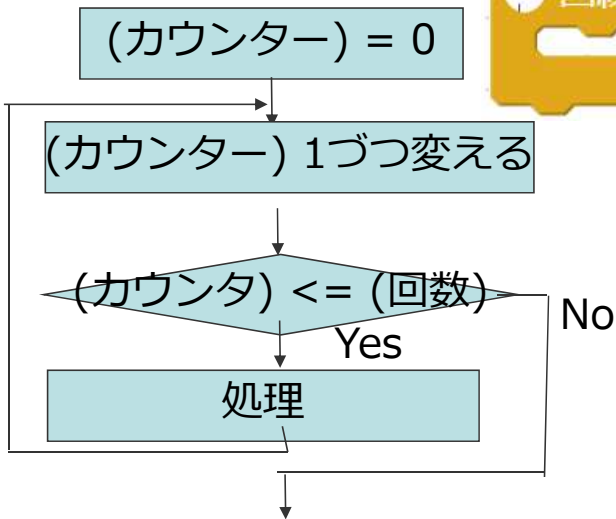


6. プログラムの構造/フローチャートの復習(2)

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



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

```
Untitled
File Edit Format Run Options Window Help
|
if x == 0:
    print("0です")
    a = x
else:
    print("0以外の数です")
    b = x
print(x)
Ln: 1 Col: 0
```

```
Untitled
File Edit Format Run Options Window Help
|
if x == 0:
    print("0です")
else:
    if x > 0:
        print("正の数です")
    else:
        print("負の数です")
Ln: 1 Col: 0
```

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
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。



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ず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ず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。

ヒント1: 次のスライド見て下さい。

チャレンジb

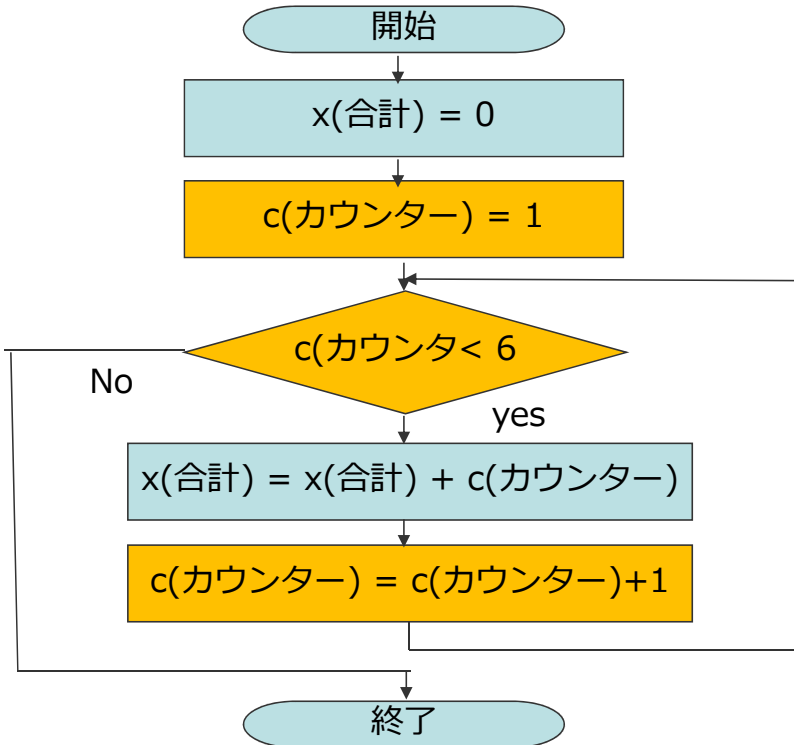
2から 10までの偶数の合計をfor文を使って計算するプログラムを作ってください。

ヒント2: チャレンジaとチャレンジbのプログラムは、ほぼ同じです。一つ前のスライドのrange()の使い方をよく見て下さい。

10. チャレンジ: 1 から 10までの合計:ヒント

チャレンジa

1 から 10までの合計を計算するフローチャート



Pythonプログラム

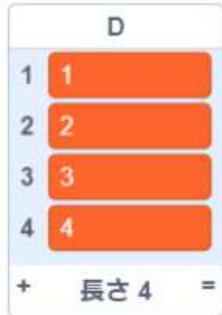
```
for c in range(1, 6):
```

左のフローチャートの黄色い部分と同等の動作を上の一行為のpythonプログラムでやっています。

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入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。

補足: print()も関数で、引数の内容を画面に表示

```
print("Hello")
```

```
Hello
```

'Helloという文字列を表示しています。

```
a = 10
```

```
print(a)
```

```
10
```

'aという変数を引き渡してして、その内容を表示。

```
print( 1+2)
```

```
3
```

'1+2の計算結果を引き渡して、3を表示。

```
d = [10, 4, 6, 5, 8]
```

```
print( d)
```

```
[10, 4, 2, 5, 8]
```

'リスト全体のdを引き渡して、その全体を表示。

```
d = [10, 4, 6, 5, 8]
```

```
print( d[2])
```

```
6
```

'リストのd[2] (d[0]=10, d[1]=4...)の値を表示

12. コードウィンドウとシェルウィンドウ

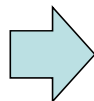
コードウィンドウにプログラムを入力して実行してみよう。
その後、シェルウィンドウで状態を確認しよう。

コードウィンドウ

確認・実行

シェルウィンドウ

```
Untitled
File Edit Format Run Options Window Help
D = [1,3,5,7,9]
for x in D:
    print(x)
Ln: 1 Col: 0
```



```
>>> 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。

次から2スライドで
ヒント1, ヒント2があります。

13. チャレンジ: リストの中の数の合計(ヒント1)

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:
```

補足:forで使うrange()の意味

```
d = list(range(1,11)) 'rangeで作った数列をリスト化
```

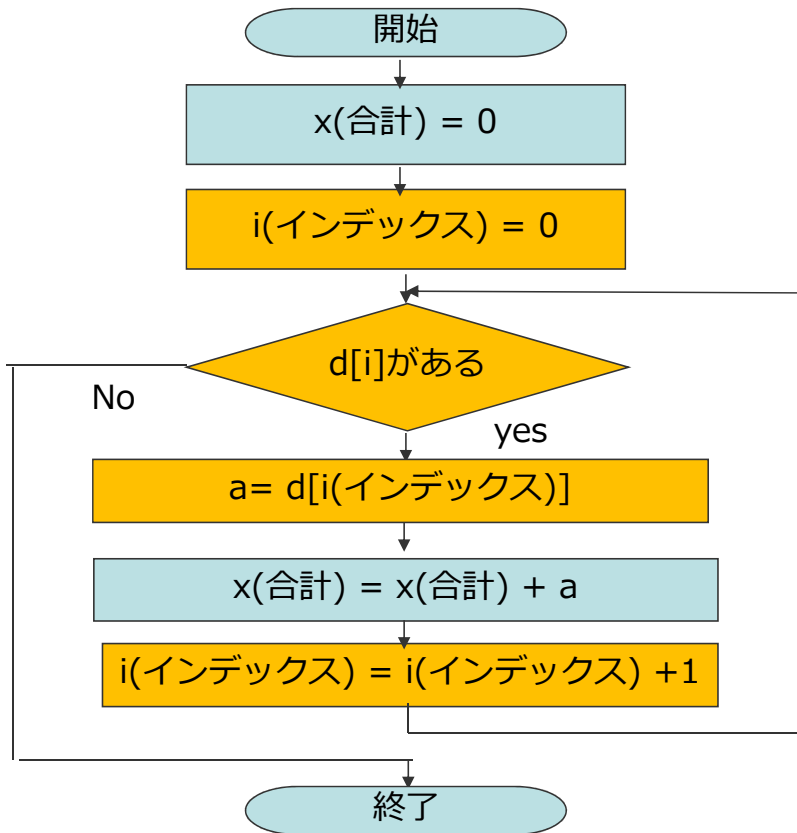
```
print(d)
```

```
[1,2,3,4,5,6,7,8,9,10]
```

```
for in d: '同じ for in range(1,11)
```

13. チャレンジ: リストの中の数の合計(ヒント2)

リストd[10, 20, 30, 40, 50, 100]の合計を計算する



Pythonプログラム

for a in d:

左のフローチャートの黄色い部分と同等の動作を上の一
行のpythonプログラムでやっています。

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す。関数はExcelの=SUM()や=Average()を自分で作る感じ
です。

確認・実行

```
def Tasu(x, y):  
    kekka = x + y  
    return kekka  
  
a = 10  
b = 20  
c = Tasu(a, b)  
print(c)
```

Ln: 1 Col: 0

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。

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

Python

確認・実行



終了の時の条件を指定

```
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)

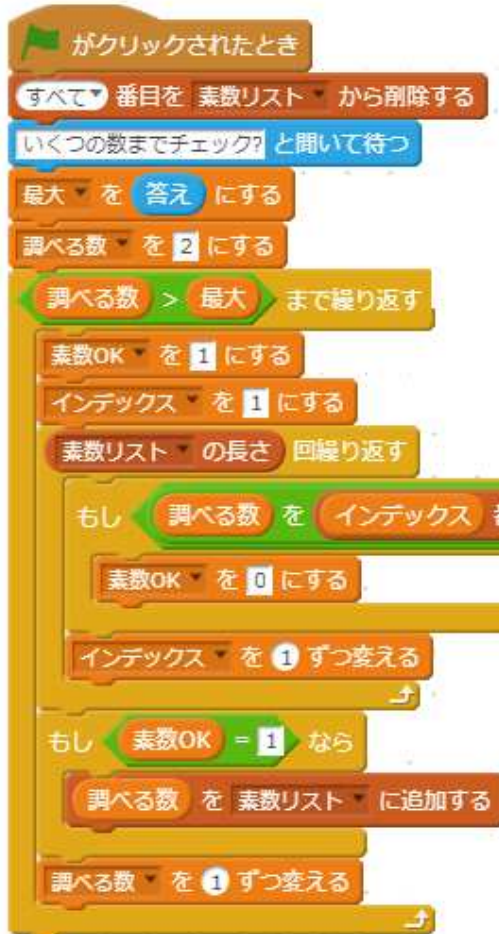
Pythonのプログラムを実行して、何をしているか確認してみよう

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。

チャレンジ:素数を求める(割り算方式)

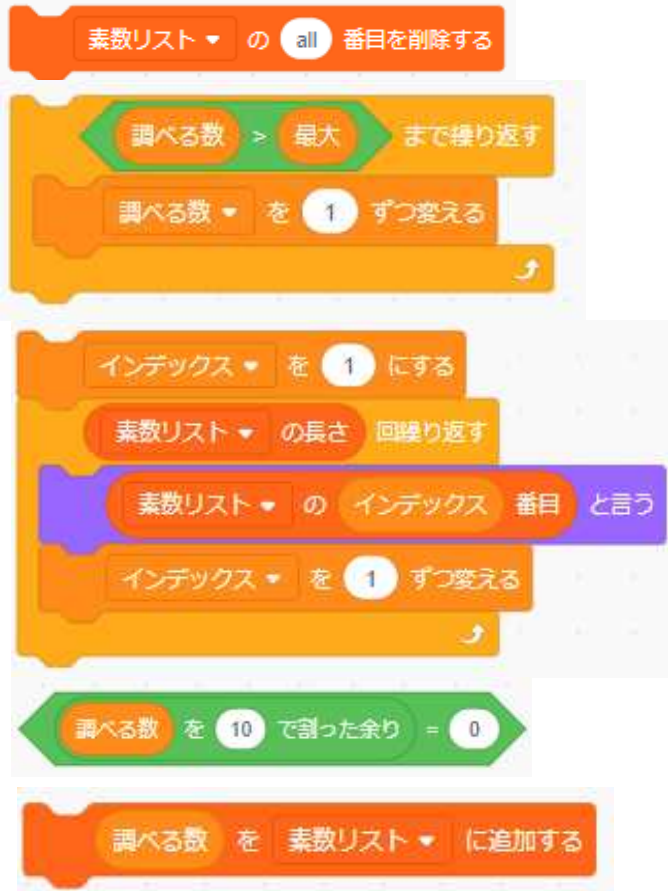


Scratchのプログラムの場合と
考え方は同じですが、Python
の場合はもっとシンプルになり
ます。たぶん11行ぐらい

次のスライドに
ヒントがあります。

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

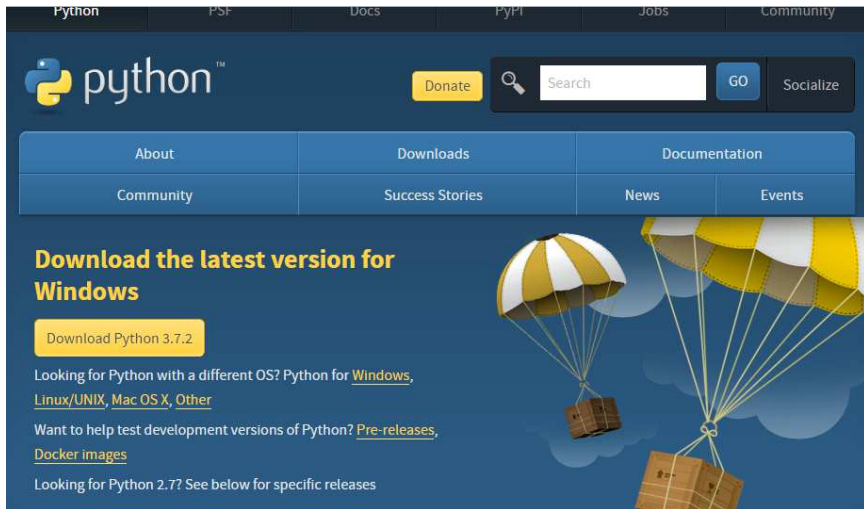
i % j == 0 '余りは%

sosu.append(i)
'リストに追加する場合

付録: Pythonのインストール

<https://www.python.org/downloads/>

からダウンロードしてインストールしてみてください。



参考: 検索するといろいろな解説したページがあります。

検索

Python 3.0 インストール