

情報科教諭向け Pythonプログラミング基礎

～ Python 実習 ～

Python Tutorとは？

Python Tutorとは？

Pythonの実行結果をインターネット上で確認することができるWebサイトです。

PC内に、Pythonの環境を作成することなく、画像付きで実行結果を確認することができます。

会員登録なども不要で、すぐに利用できます。
学生の予習・復習に活用することもできます。

Python Tutorを使ってみよう

Python Tutorを使ってみよう

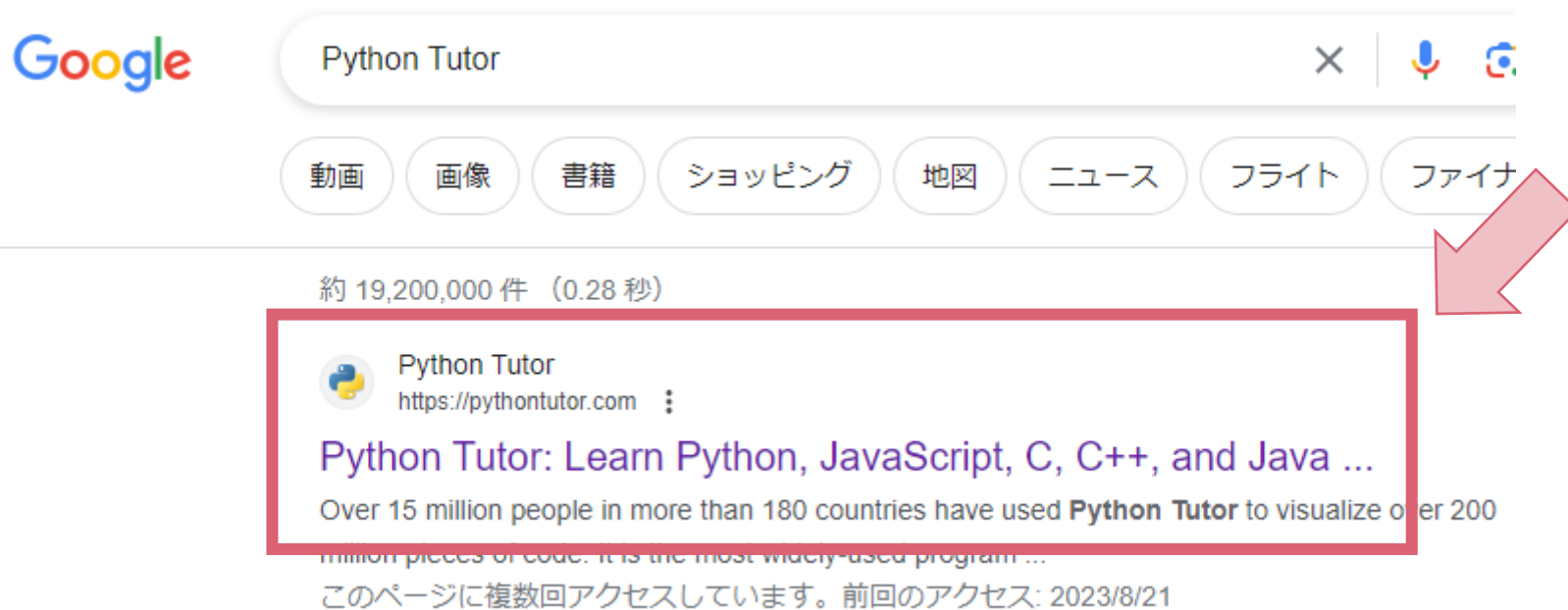
前提条件と起動方法について

- インターネットに接続されたPCであること。
- 学校内の環境で Python Tutor へのアクセスが許可されていること。アクセスができない場合は、管理者の方に下記のURLのアクセス許可を申請してください。

使用Webサイト <https://pythontutor.com/>

Python Tutorを使ってみよう

1. Google Chrome  や Microsoft Edge  などの Webブラウザを起動して、「Python Tutor」と検索



Python Tutorを使ってみよう

2. 「Start coding now in [Python](#)」をクリック

Learn Python, JavaScript, C, C++, and Java

This tool helps you learn Python, JavaScript, C, C++, and Java programming by [visualizing code execution](#).
You can use it to debug your homework assignments and as a supplement to online coding tutorials.

Start coding now in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

Over 15 million people in more than 180 countries have used Python Tutor to visualize over 200 million pieces of code. It is the most widely-used program visualization tool for computing education.

Python Tutorを使ってみよう

3. Python Tutorが準備できました。

Write code in に 「 Python 3.6 」 と表示されていることを確認


Online Python compiler and debugger - Python Tutor

Write code in Python 3.6

1

操作方法について

操作方法について



Online Python compiler and debugger - Python Tutor - Learn Python by visualizing code (also debug [Java](#), [C](#), and [C++](#) code)

Write code in Python 3.6

1

実行したいプログラムを入力

Visualize Execution ボタンをクリックして
実行する

NEW: if you use ChatGPT or other AI, [take this short survey](#)

hide exited frames [default] inline primitives, don't nest objects [default] draw pointers as arrows [default]

[Show code examples](#)

使用しない

The image is a screenshot of the Python Tutor website. It features a text input area for code, a 'Visualize Execution' button, and various configuration options. Annotations include a red arrow pointing to the top right, a yellow arrow pointing to the 'Visualize Execution' button, and a blue arrow pointing to the 'Show code examples' link. A large pink box highlights the code input area with the text '実行したいプログラムを入力' (Enter the program you want to run).

操作方法について

Write code in Python 3.6

```
1 score = 100
2 print(score)
```

以下の2行を入力

```
score = 100
print(score)
```

★文字の表示(出力)を表すprintの
pは「小文字」で入力が必要

Visualize Execution
ボタンをクリックして
実行する

Visualize Execution

操作方法について

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

プログラムを表示

→ いま実行した行

→ 実行する次の行

Python 3.6
([known limitations](#))

```
→ 1 score = 100  
   2 print(score)
```

[Edit this code](#)

→ line that just executed

→ next line to execute



<< First < Prev Next > Last >>

Step 1 of 2

[Customize visualization](#)

Print output (drag lower right corner to resize)

Frames

Objects

実行結果を表示

Next > ボタンで 1行ずつ実行

Last >> ボタンで プログラムをすべて実行

<Prev ボタンで 1行前の状態に戻す

<<First ボタンで プログラムの実行前に戻す

操作方法について

プログラムを表示
→ いま実行した行
→ 実行する次の行

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6
(known limitations)

```
→ 1 score = 100  
2 print(score)
```

[Edit this code](#)

Print output (drag lower right corner to resize)

Frames Objects

実行結果を表示

Step 1 of 2

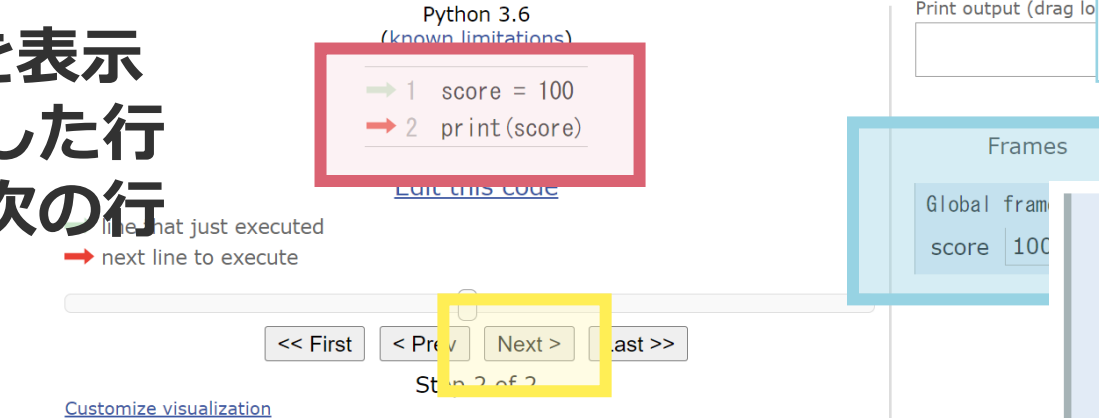
[Customize visualization](#)

Windows taskbar: 14:28 2022/08/01

操作方法について

プログラムを表示
→ いま実行した行
→ 実行する次の行

Python Tutor: Visualize code in Python, JavaScript



score = 100
scoreの変数に
100を代入する

Global frame
score 100



操作方法について

プログラムを表示
→ いま実行した行
→ 実行する次の行

The screenshot shows the Python Tutor code visualizer interface. The browser address bar displays `https://pythontutor.com/render.html#mode=display`. The page title is "Python Tutor: Visualize code in Python, JavaScript". The code editor shows the following Python code:

```
Python 3.6 (known limitations)
1 score = 100
2 print(score)
```

The second line, `print(score)`, is highlighted with a green arrow, indicating it is the next line to execute. Below the code editor, there are navigation buttons: `<< First`, `< Prev`, `Next >`, and `Last >>`. A status bar at the bottom indicates "Done running (2 steps)".

On the right side, the "Print output" panel shows the output of the program: `100`. A large blue arrow points from the `print(score)` line in the code editor to the output panel.

At the bottom of the browser window, the Windows taskbar is visible, showing the Start button, task view, and several application icons. The system clock shows 14:29 on 2022/08/01.

`print(score)`
score の値を出力

Print output

100

print関数

文字の出力はprint

```
print('Hello')
```

文字を出力するときには 小文字で print と入力

表示したい文字列は " で囲む必要がある

(Pythonでは """ でも使用可能だけど、

実教出版 / 東京書籍 は " で統一されている)

文字の出力はprint

これは Syntax Error (構文エラー)

書き方に誤りがあるときに発生するエラー

✖ `print(Hello')`

文字列が " で囲まれていない

文字の出力はprint

これは Name Error (関数エラー)

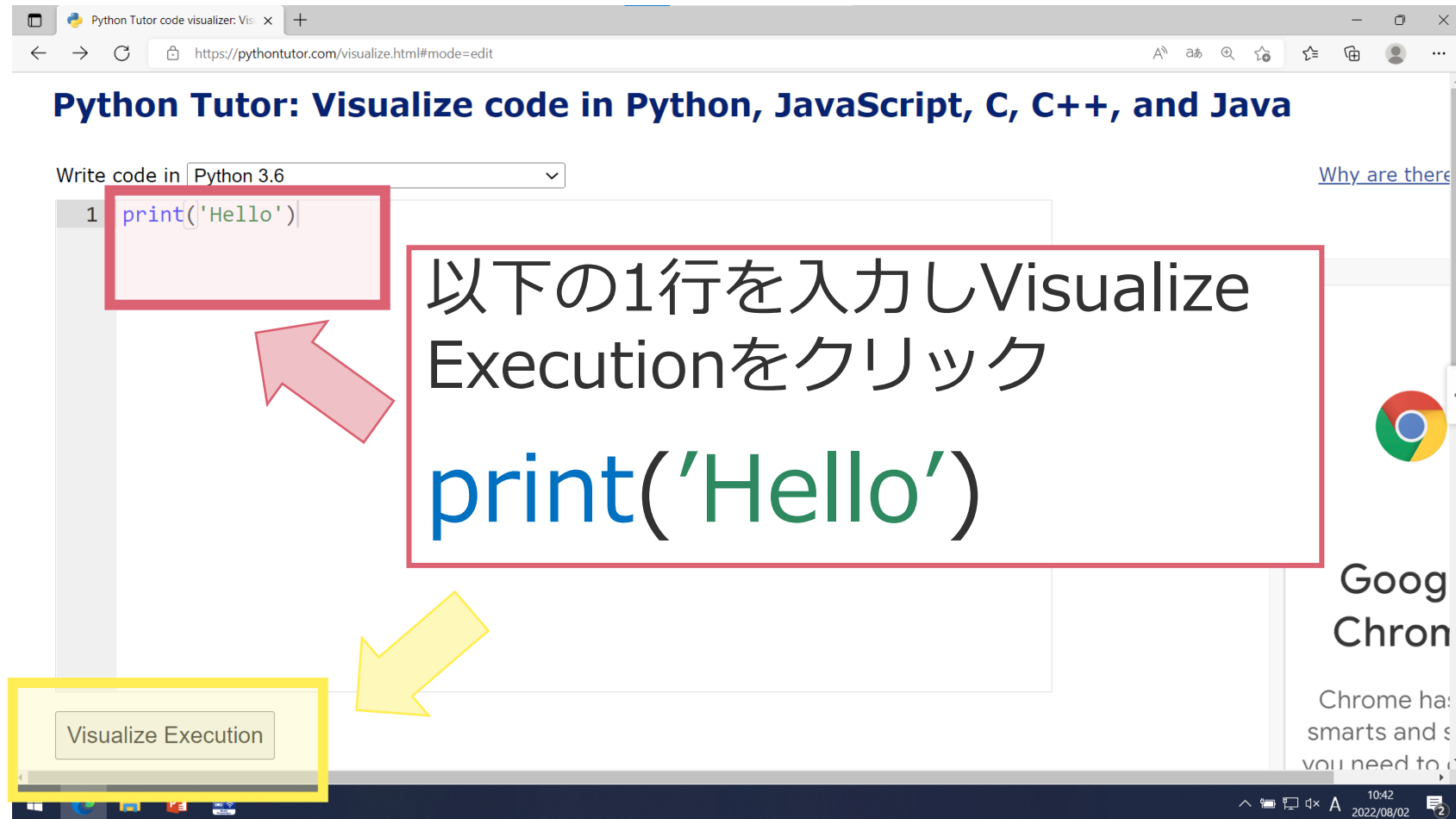
書き方に誤りがあるときに発生するエラー



Print('Hello')

print の p が大文字

文字の出力はprint



The screenshot shows the Python Tutor website interface. At the top, the title "Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java" is displayed. Below the title, there is a dropdown menu labeled "Write code in" with "Python 3.6" selected. The code editor contains a single line of code: `print('Hello')`. A red box highlights this line of code, and a red arrow points from a text box to it. The text box contains the instruction "以下の1行を入力しVisualize Executionをクリック" (Enter the following 1 line and click Visualize Execution) and the code `print('Hello')`. Below the code editor, a yellow box highlights the "Visualize Execution" button, and a yellow arrow points from the text box to it. The browser's address bar shows the URL <https://pythontutor.com/visualize.html#mode=edit>. The browser's taskbar at the bottom shows the time as 10:42 on 2022/08/02.

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Write code in Python 3.6

```
1 print('Hello')
```

以下の1行を入力しVisualize Executionをクリック

```
print('Hello')
```

Visualize Execution

文字の出力はprint

The screenshot shows the Python Tutor code visualizer interface. The browser address bar displays `https://pythontutor.com/render.html#mode=display`. The page title is "Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java". The code editor shows Python 3.6 code with a single line: `1 print('Hello')`. A red arrow points to the line number "1". Below the code editor, a legend indicates that a green arrow represents the "line that just executed" and a red arrow represents the "next line to execute". A yellow box highlights the "Next >" button in the navigation controls, with a yellow arrow pointing to it. To the right of the code editor is a "Print output" box, which is currently empty. Below the "Print output" box are tabs for "Frames" and "Objects". At the bottom of the interface, there is a "Customize visualization" link. The Windows taskbar is visible at the bottom of the screen, showing the time as 10:48 on 2022/08/02.

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6
([known limitations](#))

→ 1 `print('Hello')`

[Edit this code](#)

→ line that just executed
→ next line to execute

Next >

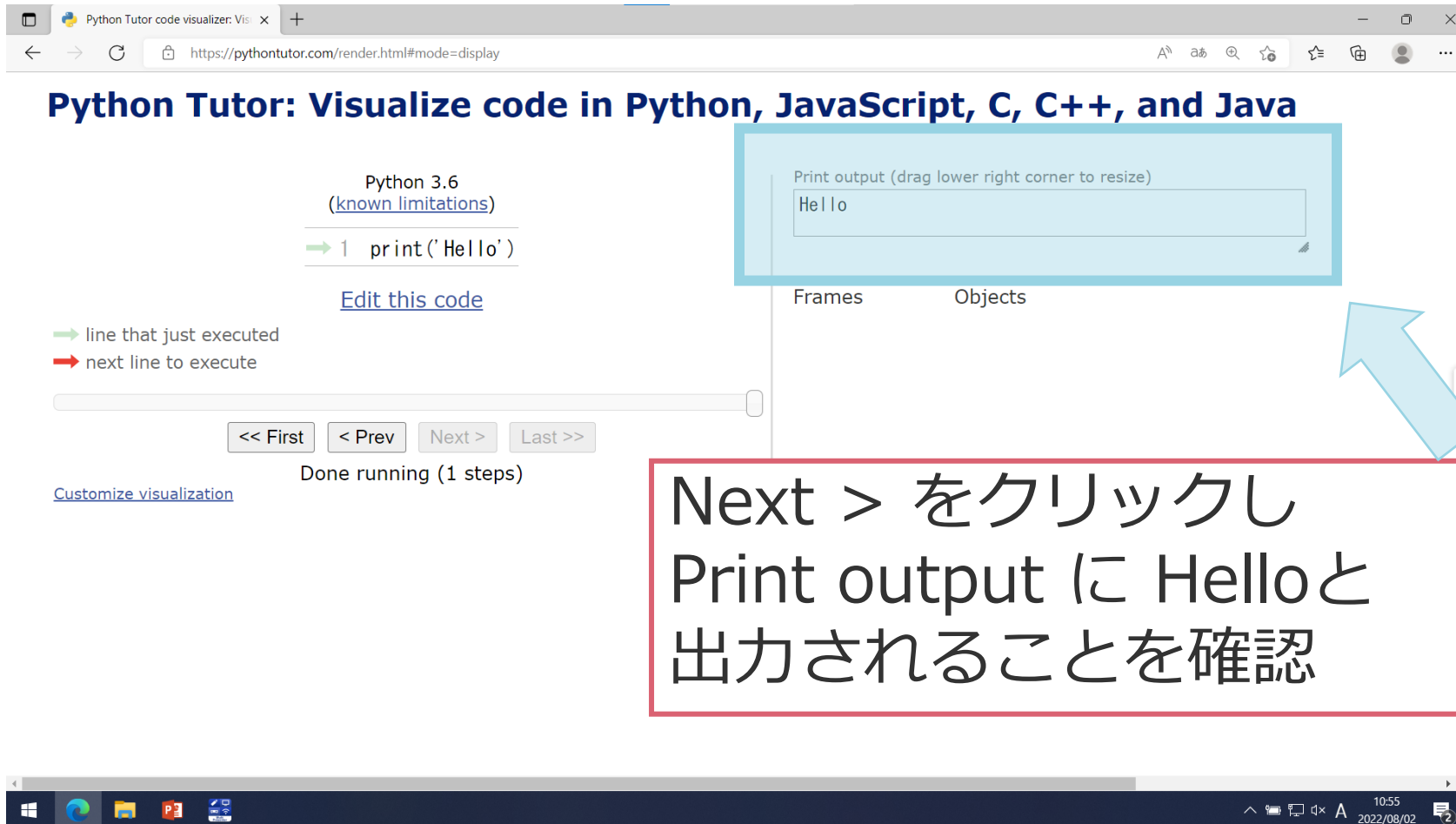
[Customize visualization](#)

Print output (drag lower right corner to resize)

Frames Objects

Next > をクリックし
Print output に Hello と
出力されることを確認

文字の出力はprint



The screenshot shows the Python Tutor code visualizer interface. The browser address bar displays `https://pythontutor.com/render.html#mode=display`. The page title is "Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java". The code editor shows Python 3.6 code with a single line: `1 print('Hello')`. A green arrow points to the line number 1, indicating it has just been executed. Below the code editor, there are navigation buttons: "<< First", "< Prev", "Next >", and "Last >>". A status bar at the bottom indicates "Done running (1 steps)". On the right side, the "Print output" window is open, showing the text "Hello". A blue arrow points from the "Next >" button to the "Print output" window. A red box with Japanese text is overlaid on the bottom right of the screenshot.

Python 3.6
([known limitations](#))

```
→ 1 print('Hello')
```

[Edit this code](#)

→ line that just executed
→ next line to execute

<< First < Prev Next > Last >>

[Customize visualization](#)

Done running (1 steps)

Print output (drag lower right corner to resize)

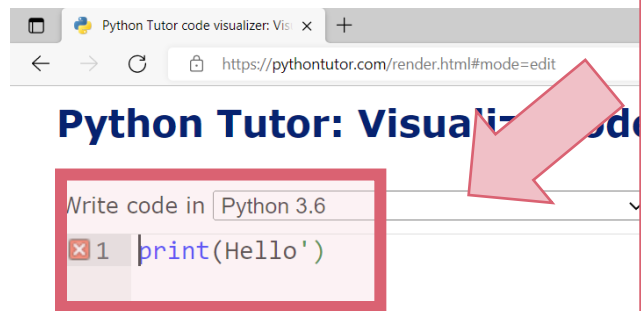
Hello

Frames Objects

Next > をクリックし
Print output に Helloと
出力されることを確認

エラーを表示しよう

エラーを表示しよう Syntax Error



以下の1行を入力しVisualize Executionをクリック

```
print(Hello')
```

SyntaxError: EOL while scanning string literal (<string>, **line 1**)
1行目の書き方に誤りがあります。



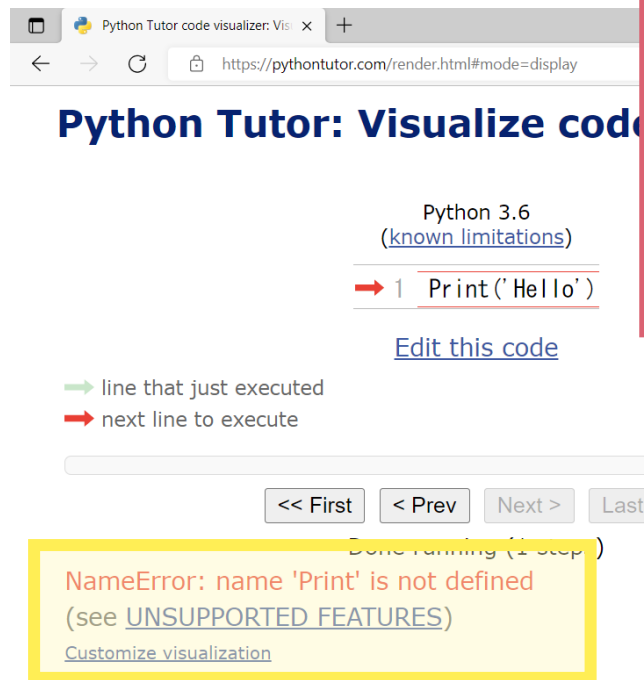
SyntaxError: EOL while scanning string literal (<string>, line 1)



Goog
Chron

Chrome has
smarts and s
you need to

エラーを表示しよう Name Error



以下の1行を入力しVisualize Executionをクリック

`Print('Hello')`

NameError: name 'Print' is not defined
'Print'という名前は定義されていません。

変数と代入について

変数に値を代入する

score = 100

season = 'summer'

変数というのは値を保存する箱のようなものです。

変数名を左側に、代入したい値を右に表します。

Global frame

score

Global frame

season

"summer"

変数に値を代入する

変数名を指定するためのルール

1. ifなどの「**予約語**」は使用できない
2. 先頭の文字に「**数字**」は許されない
3. 「**小文字を使った分かりやすい名前**」が推奨される

予約語の一覧

ifなどの「**予約語**」は変数名に利用できない

False, None, True, and, as, assert, async,
await, break, class, continue, def, del, elif,
else, except, finally, for, from, global, if,
import, in, is, lambda, nonlocal, not, or,
pass, raise, return, try, while, with, yield

変数に値を代入する

ifなどの「**予約語**」は変数名に利用できない

 `pass = 100`

先頭の文字に「**数字**」は許されない

 `1score = 100`

変数に値を代入する

score = 100

score = 80

Global frame

Global frame

score 80

変数の値を上書きする場合も、次の行に同じように入力すると値を上書きすることができる

変数に値を代入する

これはIndentation Error



```
score = 100
```

```
↔ score = 80
```

Pythonではインデントがとても大切

変数に値を代入する

この書き方は推奨しない

score = 100

score = 'pass'



Global frame

s

Global frame

score "pass"

変数に値を代入する

cnt = 1

cnt = cnt + 1

Global frame

Global frame

cnt 2

変数の値にさらに 1 を追加することもできる

確認問題1

確認問題 1

1. 変数名 color に 「 red 」 を代入する
2. 変数名 color を print関数 で出力する
3. 変数名 color を 「 green 」 に変更する
4. 変数名 color を print関数 で出力する

確認問題1

```
color = 'red'
```

```
print(color)
```

```
color = 'green'
```

```
print(color)
```

変数と四則演算

変数と四則演算

Pythonでの四則演算は以下の半角記号を使用します。

加算	+
減算	-
乗算	*
除算	/
剰余	%
べき乗	**

su1	+	su2	su1	%	su2
su1	-	su2	su1	**	su2
su1	*	su2			
su1	/	su2			

変数と四則演算

加算・減算の場合

score_1 = 15

score_2 = 5

result_add = score_1 + score_2

result_sub = score_1 - score_2

Global frame

score_1	15
score_2	5
result_add	20
result_sub	10

変数と四則演算

乗算・除算の場合

```
score_1 = 15
```

```
score_2 = 5
```

```
result_mul = score_1 * score_2
```

```
result_div = score_1 / score_2
```

Global frame

score_1	15
---------	----

score_2	5
---------	---

result_mul	75
------------	----

result_div	3.0
------------	-----

確認問題2

確認問題 2 - 1

1. 変数名 price に 「 100 」 を代入する
2. 変数名 tax に price の消費税(10%)を求める計算式を設定する
3. 変数名 tax を print関数 で出力する

確認問題 2 - 1

```
price = 100
```

```
tax = price * 0.1
```

```
print(tax)
```

確認問題 2 - 2

1. 変数名 price に 「 100 」 を代入する
2. 変数名 tax に 「 1.1 」 を代入する
3. 変数名 total に 消費税込金額を求める
計算式を設定する
4. 変数名 total を print関数 で出力する

確認問題 2 - 2

price = 100

tax = 1.1

total = price * tax

print(total)

printの組み合わせ

printの組み合わせ

確認問題2を下記のように修正して実行する

```
price = 100
```

```
tax = 1.1
```

```
total = price * tax
```

```
print('TAX IN ', total)
```

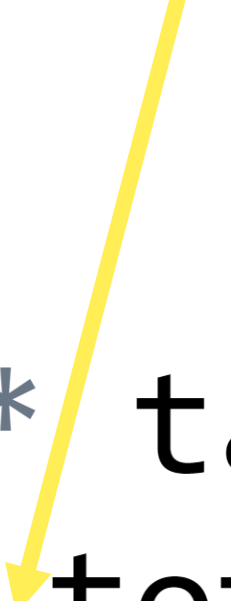
カンマで区切ると文字と
数字をprintで出力可能

printの組み合わせ

カンマで区切ると文字と数字をprintで出力可能

確認問題2を下記のように修正して実行する

```
1 price = 100
2 tax = 1.1
3 total = price * tax
4 print('tax in', total)
```



printの組み合わせ

小数点の誤差が発生する

Frames

Global frame	
price	100
tax	1.1
total	110

Print output (drag lower right cc

tax in 110.000000000000001

```
3 total = price * tax
4 print('tax in', total)
```

printの組み合わせ

確認問題2を下記のように修正して実行する

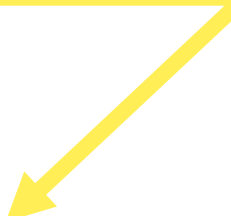
```
price = 100
```

```
tax = 1.1
```

```
total = price * tax
```

```
print(' tax in ' , '{:.2f}' .format(total) )
```

'{:.2f}' . format(値)
少数第 2 位まで表示



printの組み合わせ

確認問題2を下記のように修正して実行する

```
1 price = 100
2 tax = 1.1
3 total = price * tax
4 print('tax in', '{:.2f}'.format(total))
```

確認問題3

確認問題 3 - 1

1. 変数名 price に 「 555 」 を代入する
2. 変数名 tax に 「 1.1 」 を代入する
3. 変数名 total に 消費税込金額を求める
計算式を設定する
4. 'tax in' という文字と 変数名 total(小数点
以下切り捨て) を表示する

確認問題 3 - 1

price = 555

tax = 1.1

total = price * tax

print(' tax in ' , '{:.0f} ' .format(total))

確認問題 3 - 2

1. 変数名 height に 「 170 」 を代入する
2. 変数名 weight に 「 61 」 を代入する
3. 変数名 BMI に体重 kg \div (身長 m)²で計算した値を代入する
4. 変数名 BMI を print関数 で出力する

確認問題 3 - 2

height = 170

weight = 61

BMI = 61 / (170 / 100)**2

print(BMI)

コメント

コメント

消費税率は10%

print(sum * 1.1)

ソースコード内に、#を半角記号で入力すると、その行までの終わりがコメントとなる

コメントは、プログラムの**実行に影響を与えない**

コメント

より右側はコメント

```
# print(sum * 1.08)
```

```
print(sum * 1.1) # 消費税
```

実行したくないプログラムをコメントにすることで、消さずにソースコード内に残すことも可能

if 条件分岐

if 条件分岐

== x と y が等しい
!= x と y が等しくない

条件式によってプログラムを分岐したいときに if を使用する

条件の書き方は、以下の通り

$x < y$, $x \leq y$, $x > y$, $x \geq y$

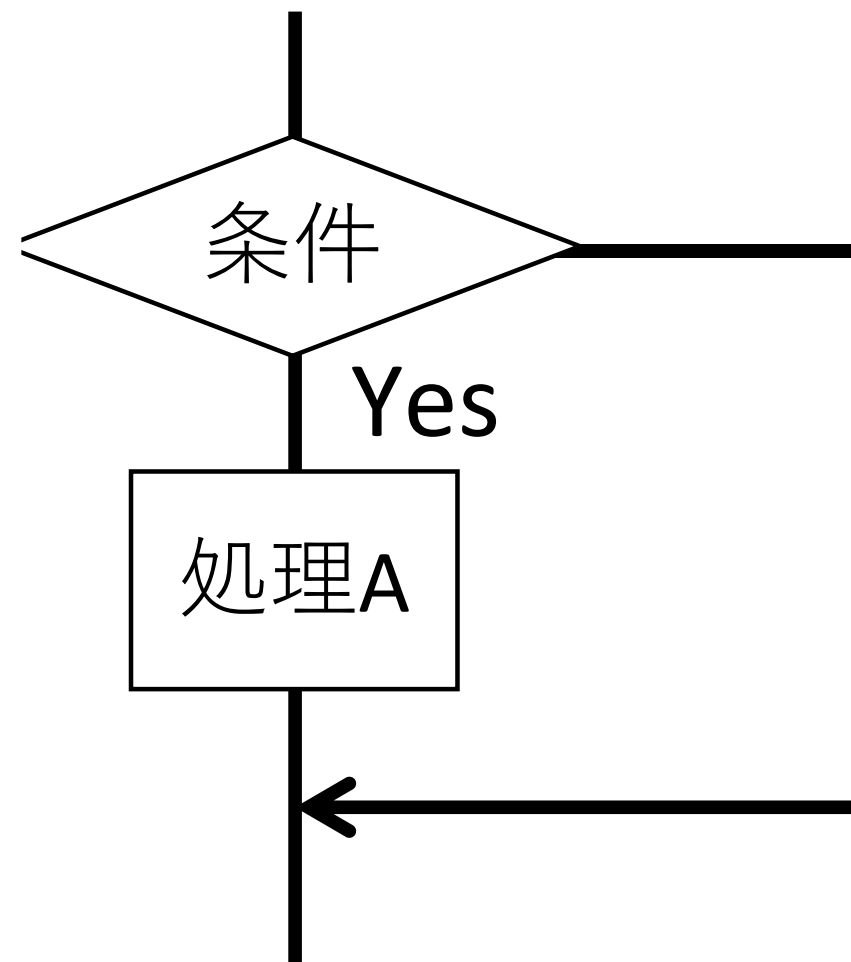
$x == y$, $x != y$

if 条件分岐

if 条件：

Yesの時

処理A



if 条件分岐

これは Indentation Error (インデントエラー)

if 条件 :

Yesの時

処理A

ifの行の次には1文字以上の
インデントが必要
(一般的には4文字分の
半角スペースを入力)

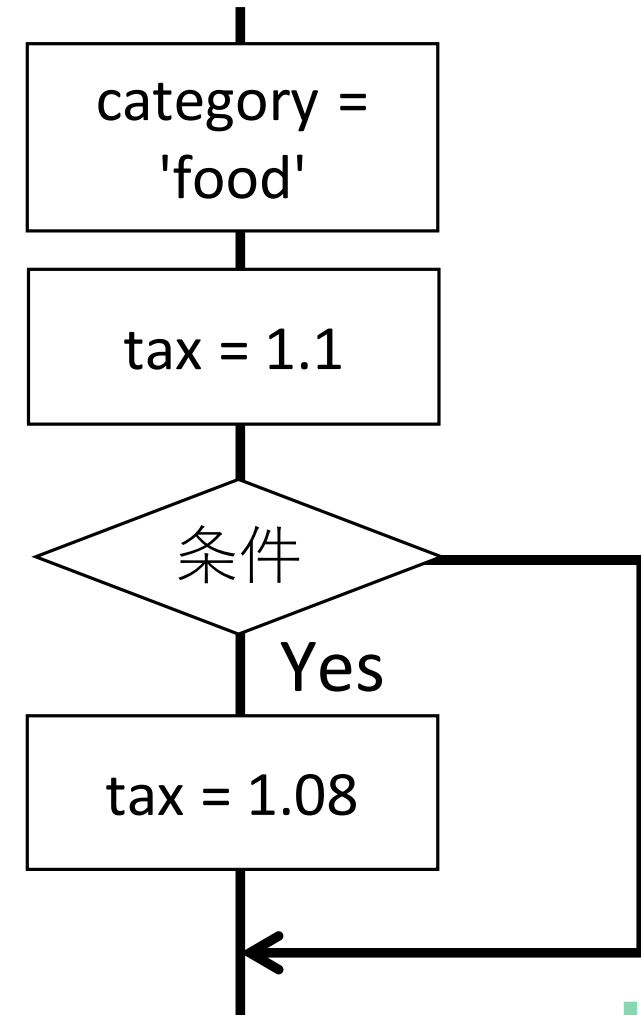
if 条件分岐

```
category = 'food'  
tax = 1.1
```

```
if category == 'food':
```

```
    # Yesの時  
    tax = 1.08
```

```
print('tax: ', tax )
```



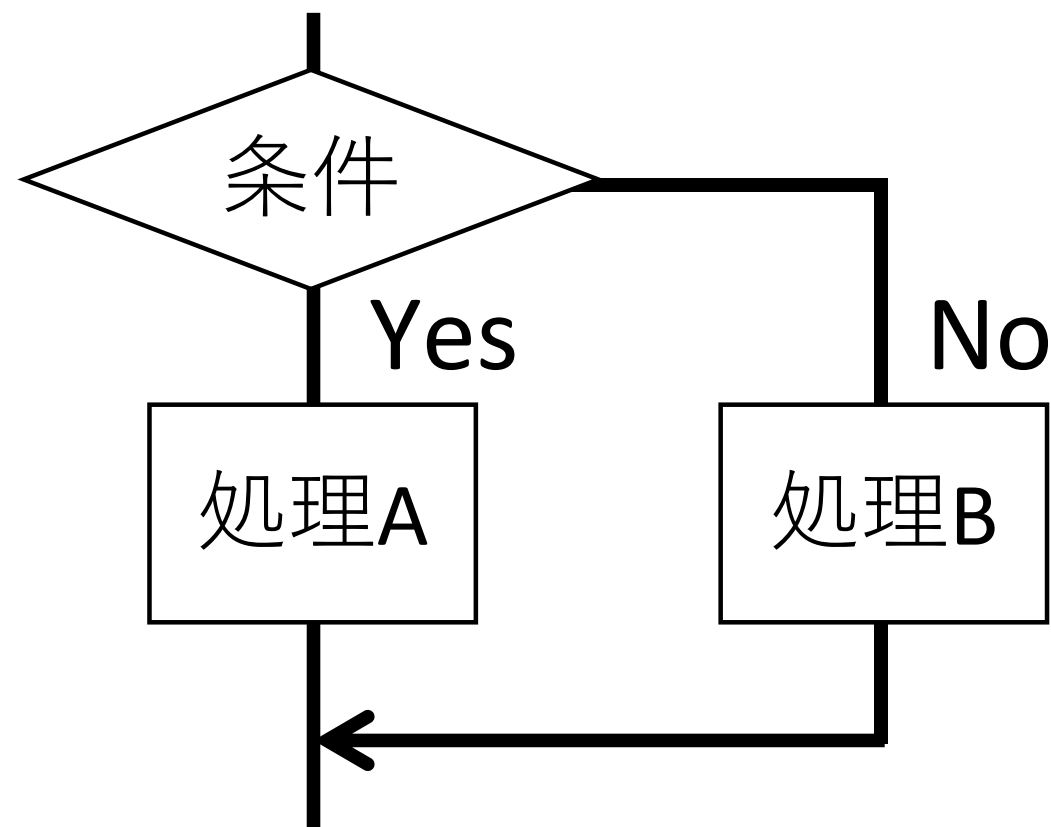
if 条件分岐

if 条件：

Yesの時
処理A

else:

Noの時
処理B

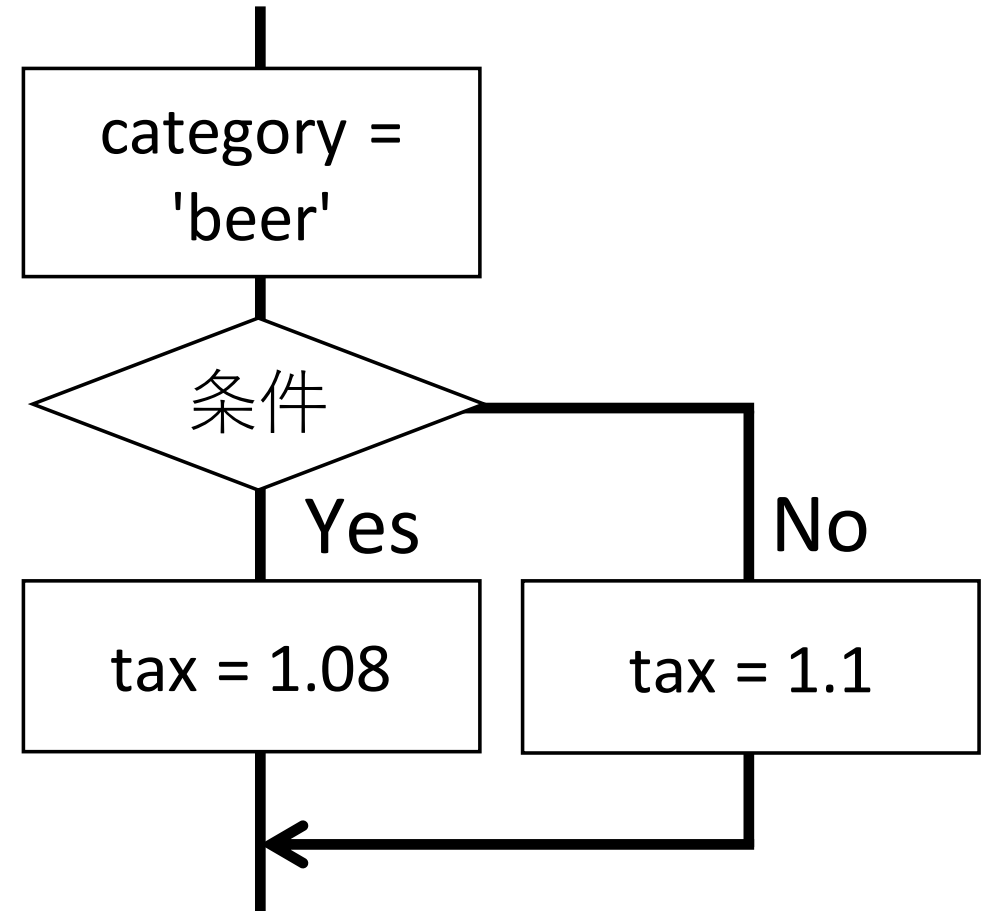


if 条件分岐

```
category = 'beer'

if category == 'food':
    tax = 1.08 #Yesの時
else:
    tax = 1.1 #Noの時

print('tax: ', tax )
```



if 条件分岐

if 条件1 :

処理A # 条件1がYes

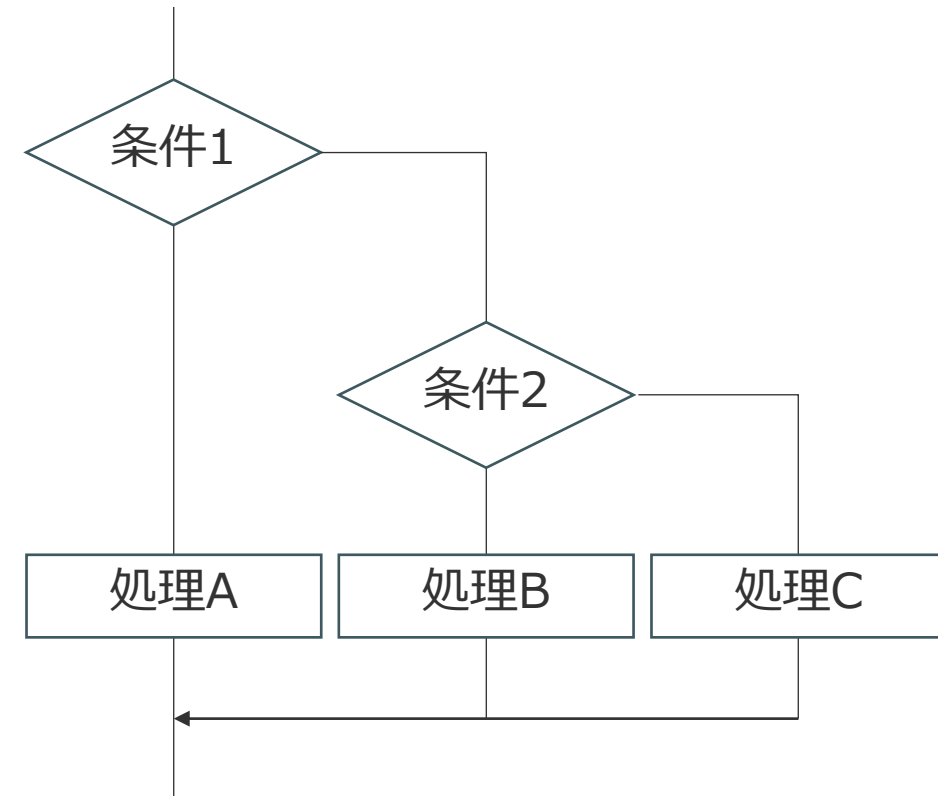
else:

if 条件2 :

処理B # 条件1がNo,条件2がYes

else :

処理C # 条件1も条件2もNo



**次のフリップの
書き方でもOK！**

if 条件分岐

if 条件1 :

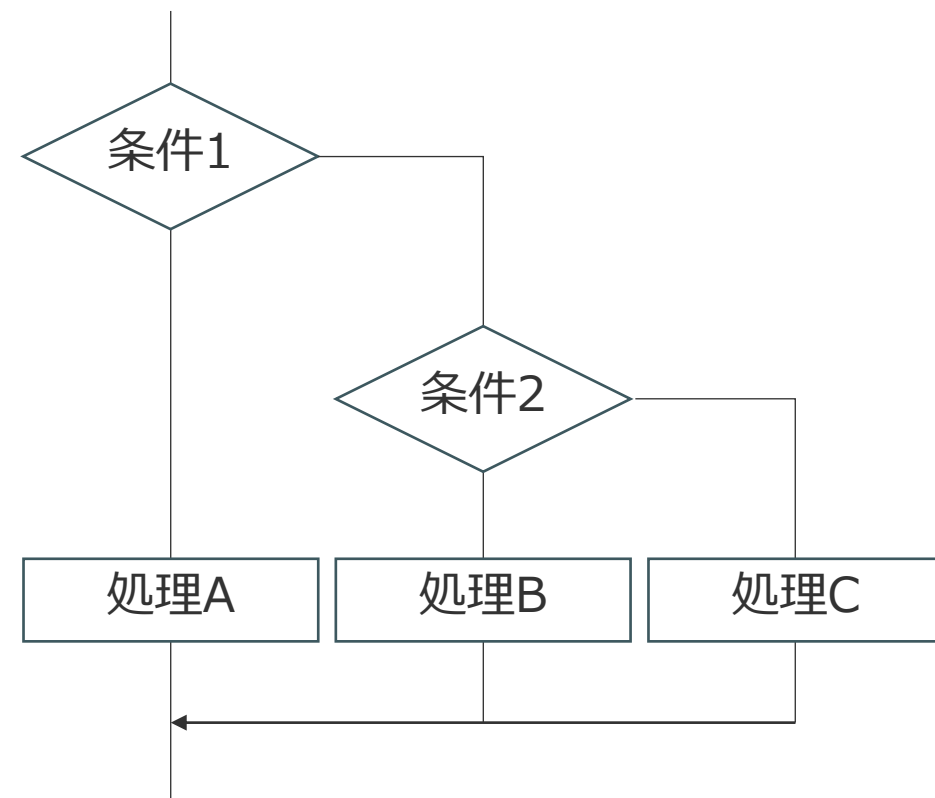
処理A # 条件1がYes

elif 条件2:

処理B # 条件1がNo,条件2がYes

else :

処理C # 条件1も条件2もNo



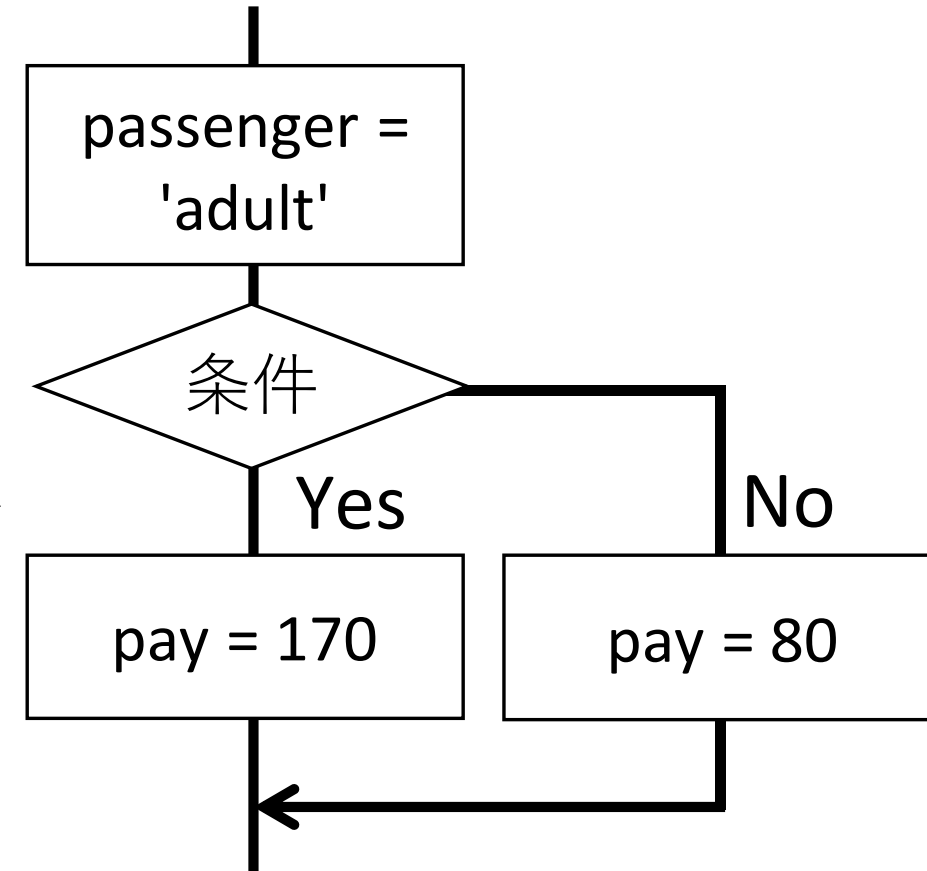
確認問題4

確認問題 4

変数名 passenger に乗客の区分を指定する

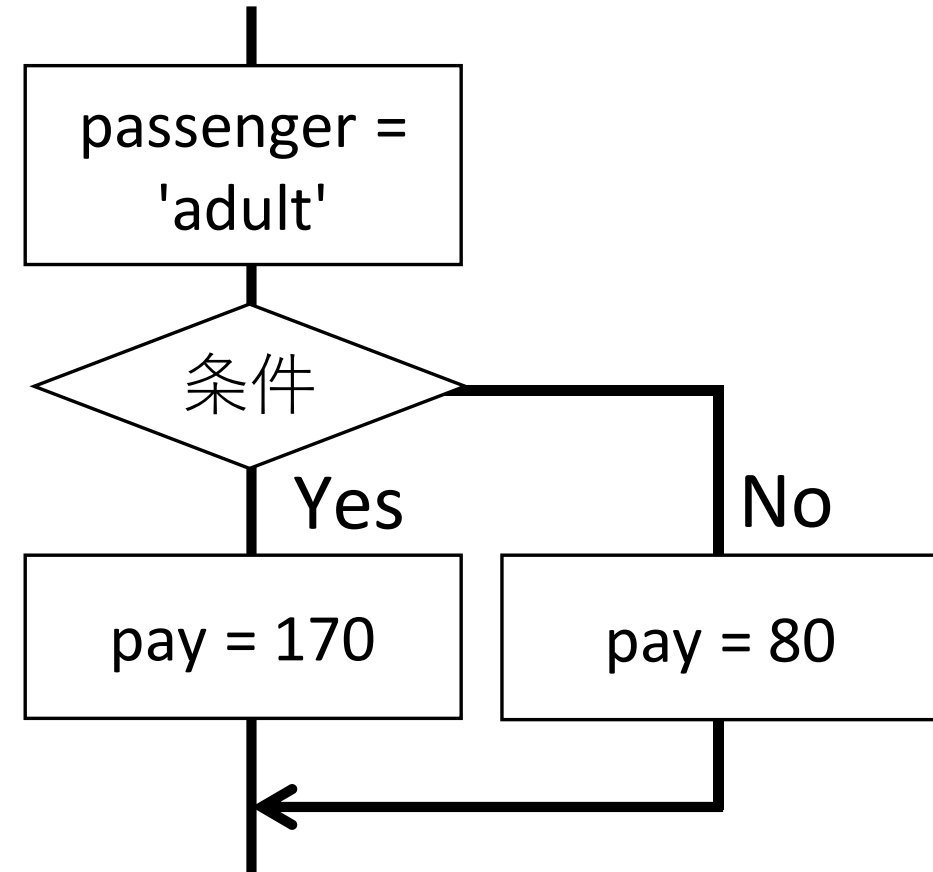
passenger に adult が設定されたときには 変数 pay に 170を代入し、それ以外ときには 変数 pay に 80を代入する

最後に pay の値を出力する



確認問題 4

```
passenger = 'adult'  
if passenger == 'adult':  
    pay = 170  
else:  
    pay = 80  
print('pay: ', pay)
```



リスト list と タプル

リスト list と タプル

```
coins = [1, 5, 10, 50, 100, 500]
```

リストとは、**複数の値を1つにまとめて管理**するためのもの

リストの要素は コンマ「,」区切りで表す

リスト list と タプル

```
coins = [1, 5, 10, 50, 100, 500]
```

```
print(coins) #リストを全て出力
```

coins	1	5	10	50	100	500
	0	1	2	3	4	5

リスト list と タプル

```
coins = [1, 5, 10, 50, 100, 500]
```

```
print(coins[0]) #0番の「1」を出力
```

リストは左側から 0 番目と数える

coins	1	5	10	50	100	500
	0	1	2	3	4	5

リスト list と タプル

```
coins = [1, 5, 10, 50, 100, 500]
```

```
print(coins[0:2]) #0以上2未満
```

スライスといった方法で範囲を絞った出力も可能

coins						
	1	5	10	50	100	500
	0	1	2	3	4	5

リスト list と タプル

```
coins = [1, 5, 10, 50, 100, 500]
```

```
print(coins[3:]) #3以上すべて
```

スライスといった方法で範囲を絞った出力も可能

coins	1	5	10	50	100	500
	0	1	2	3	4	5

リスト list と タプル

```
coins = [1, 5, 10, 50, 100, 500]
```

```
print(coins[:4]) #0から4未満
```

スライスといった方法で範囲を絞った出力も可能

coins						
	1	5	10	50	100	500
	0	1	2	3	4	5

リスト list と タプル

```
currency = ['USD', 'EUR', 'JPY']
```

リストに文字列を格納するときには " または "" で囲む

リスト名だけ は リスト内すべての値を出力

リスト名[要素の番号] で 取り出したい値のみを出力

リスト list と タプル

```
bills = [1000, 5000, 10000]
```

```
bills[0] = 2000 #0番目の値を変更
```

```
print(bills) #リストを全て出力
```

bills	2000	5000	10000
	0	1	2

リスト list と タプル

```
bills = [1000, 5000, 10000]
```

```
bills.append(2000) #末尾に挿入
```

```
print(bills) #リストを全て出力
```

bills	1000	5000	10000	2000
	0	1	2	3

リスト list と タプル

```
bills = [1000, 5000, 10000]
```

```
bills.insert(1, 2000) # 指定して挿入
```

```
print(bills) # リストを全て出力
```

bills	1000	2000	5000	10000
	0	1	2	3

リスト list と タプル

```
bills = [1000, 2000, 5000, 10000]
```

```
bills.remove(2000) #指定して削除
```

```
print(bills) #リストを全て出力
```

bills	1000	2000	5000	10000
	0	1	2	3

リスト list と タプル

```
bills = [1000, 2000, 5000, 10000]
```

```
bills.remove(2000) #指定して削除
```

```
print(bills) #リストを全て出力
```

bills	1000	5000	10000
	0	1	2

リスト list と タプル

リストに存在しないものは Value Error

```
bills = [1000, 2000, 5000, 10000]
```

 `bills.remove(50000)` #リストにない

bills	1000	2000	5000	10000
	0	1	2	3

リスト list と タプル

```
bills = [1000, 2000, 5000, 10000]
```

```
del bills[1] #1番目の2000を削除
```

```
print(bills) #リストを全て出力
```

bills	1000	5000	10000
	0	1	2

確認問題 5

確認問題 5

リスト名 `week` に以下の曜日を設定し、
リストの一覧を `print` で出力する

week	Sun	Mon	Tue	Wed	Thu	Fri	Sat
	0	1	2	3	4	5	6

確認問題 5

```
week = [ 'Sun' , 'Mon' , 'Tue' , 'Wed' ,  
         'Thu' , 'Fri' , 'Sat' ]
```

```
print(week)
```

※ サイズの関係で2行に分かれています。 1行で入力してください

確認問題 6

確認問題 6

リスト名 `week` の先頭に「every」を追加して、
「Tue」「Wed」「Thu」を print 1行 で出力る

week	every	Sun	Mon	Tue	Wed	Thu	Fri	Sat
	0	1	2	3	4	5	6	7

確認問題 6

```
week = ['Sun','Mon','Tue','Wed','Thu','Fri','Sat']
```

```
week.insert(0, 'every')
```

```
print(week[3:6])    #3以上 6未満
```

week	every	Sun	Mon	Tue	Wed	Thu	Fri	Sat
	0	1	2	3	4	5	6	7

リスト list と タプル

```
age = [3,2,6,1,7,4]
```

```
age.sort() #昇順「小さい順」
```

```
age.sort(reverse=True) #降順
```

sort は リストの要素そのものを入れ替える

リスト list と タプル

```
1 kids = [3, 2, 6, 1, 7, 4]
2 print(kids)
3 kids.sort()
4 print(kids)
5 kids.sort(reverse=True)
6 print(kids)
```

kids

7	6	4	3	2	1
0	1	2	3	4	5

リスト list と タプル

coins = (1, 5, 10, 50, 100, 500)

coins = 1, 5, 10, 50, 100, 500

タプルは、1度設定した要素を**変更することができない**

タプルの () は省略することもできる

リスト list と タプル

これは **Type Error**

()で囲まれている場合はタプル

coins = (1, 5, 10, 50, 100, 500)



coins[0] = 3

タプルは、1度設定した要素を変更することができない

while 繰り返し

while 繰り返し

while 条件 :

↔ 繰り返し処理

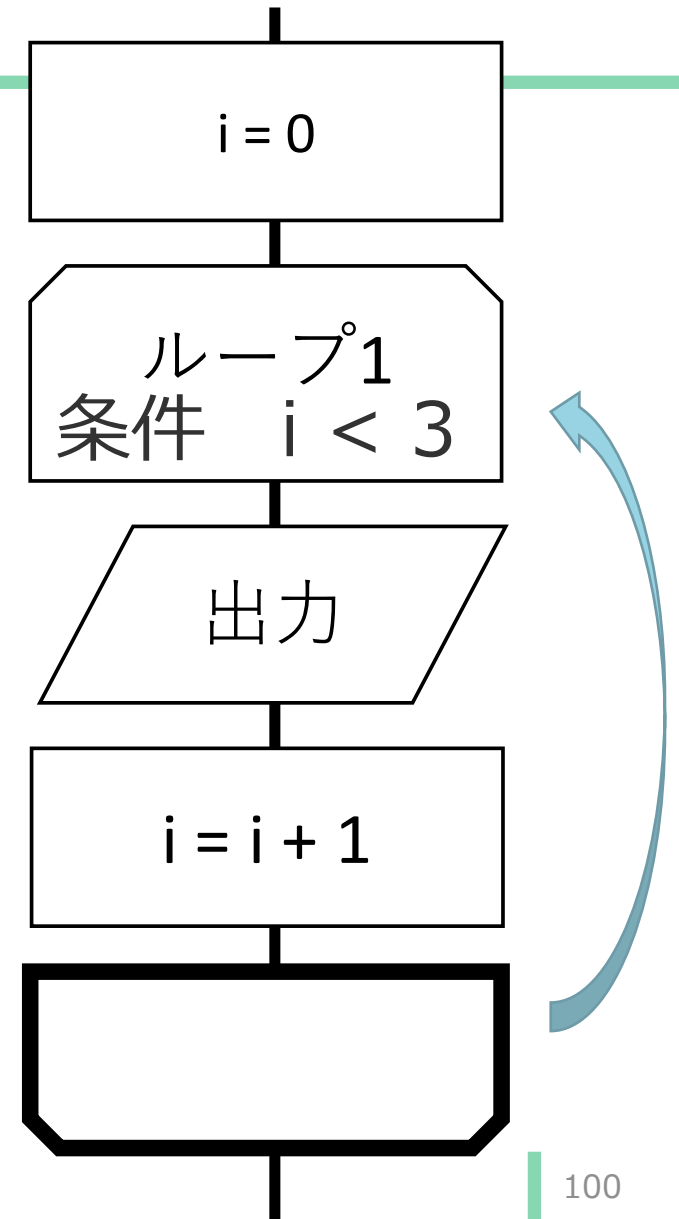
繰り返し処理を行いたいときにwhileも使用できる

while 繰り返し

```
i = 0  
while i < 3 :
```

```
    print ( i )  
    i = i + 1
```

繰り返したい処理



while 繰り返し

```
1 i = 0
2 while i < 3:
3     print(i)
4     i = i + 1
```

Print output

```
0
1
2
```

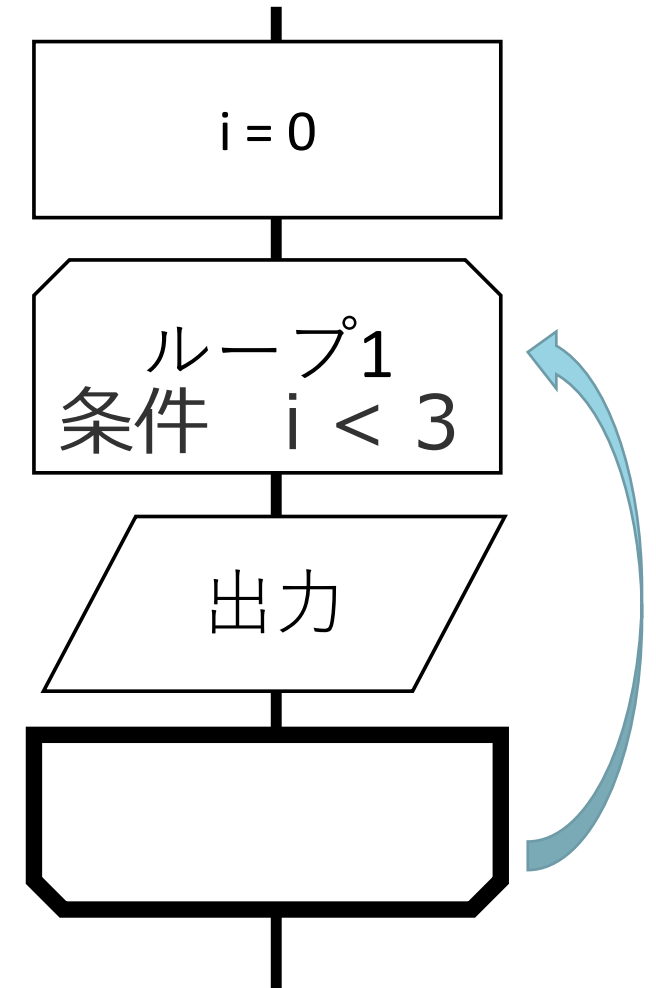
while 繰り返し

`i = 0`

`while i < 3 :`

`print (i)`

繰り返したい処理



while 繰り返し

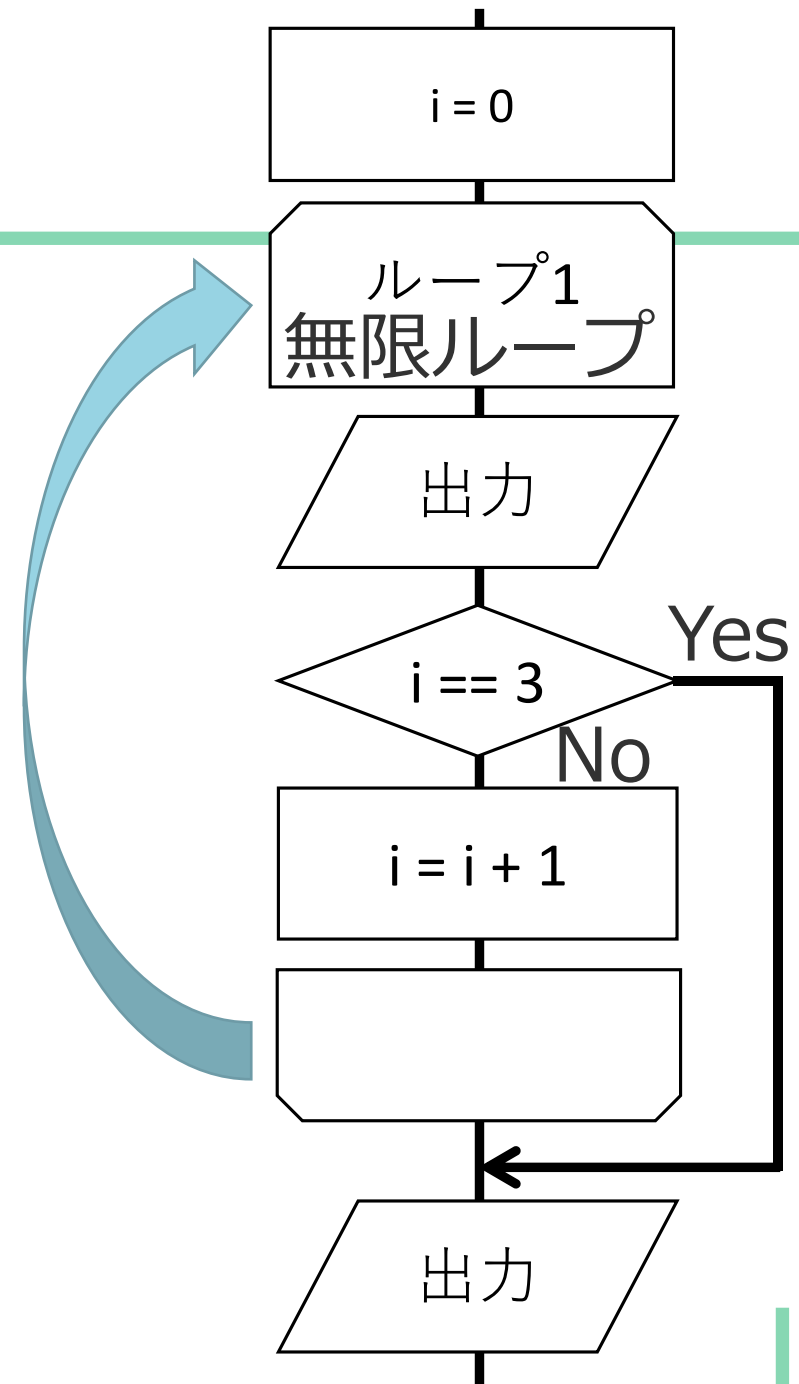
```
1 i = 0
2 while i < 3:
3     print(i)
```

Print output (draw)

0
0
0
0

while 繰り返し

```
i = 0
while True:
    print(i)
    if i == 3:
        break
    i = i + 1
print('FIN')
```



for 繰り返し

for 繰り返し



The diagram illustrates the syntax of a for loop. The text "for 変数 in オブジェクト :" is shown. "for" is in blue, "変数" is underlined in blue, "in" is in blue, and "オブジェクト" is enclosed in a red box. A blue curved arrow points from "オブジェクト" to "変数". Below this, "繰り返し処理" is enclosed in a yellow box. A red double-headed arrow points from the left side of the slide to the yellow box. A vertical red dashed line is on the far left.

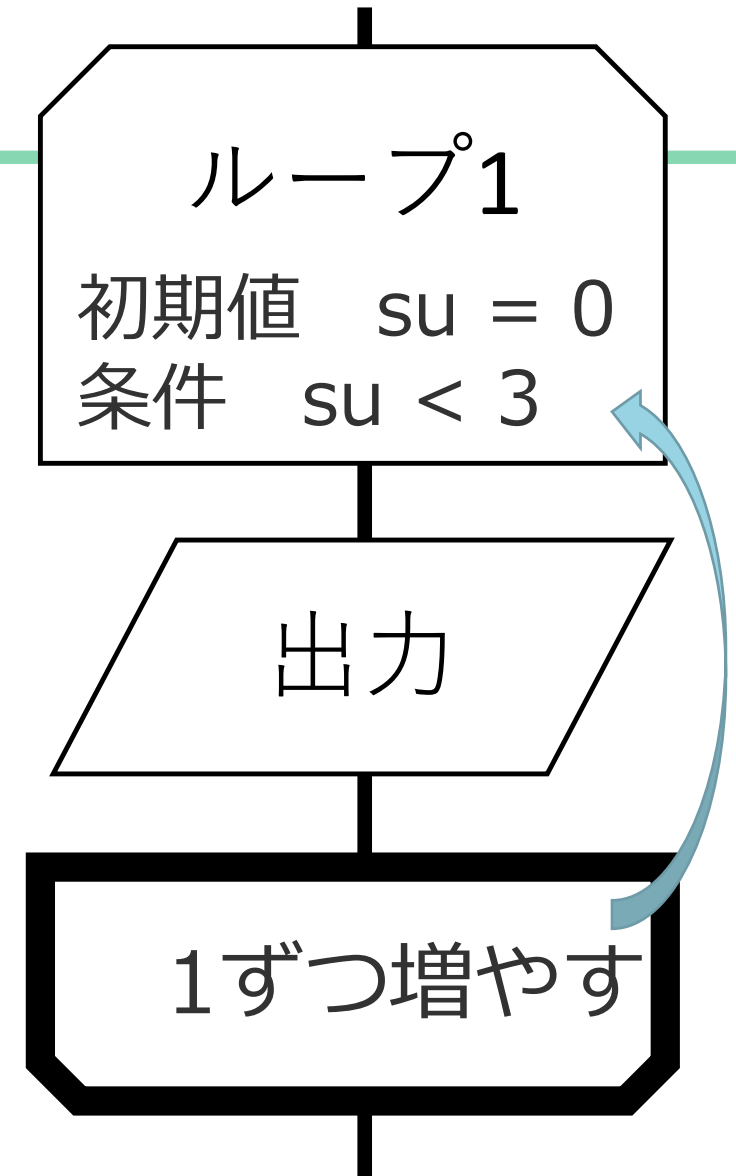
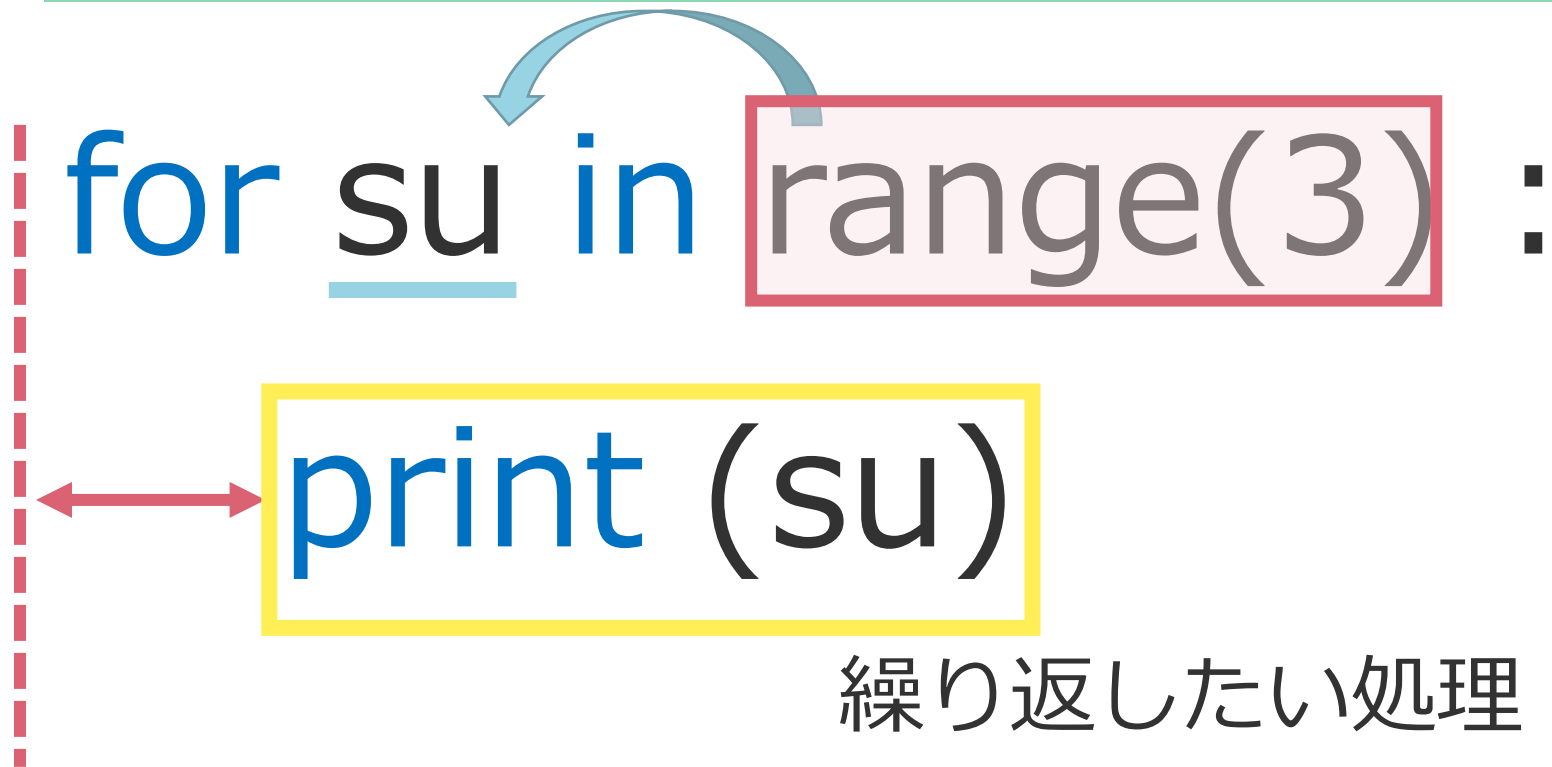
```
for 変数 in オブジェクト :  
    繰り返し処理
```

繰り返し処理を行いたいときにforを使用する

(オブジェクトの数だけ実行したいとき)

for 繰り返し

`for su in range(3) :`
`print (su)`
繰り返したい処理



for 繰り返し

```
1 for su in range(3):  
2     print(su)
```

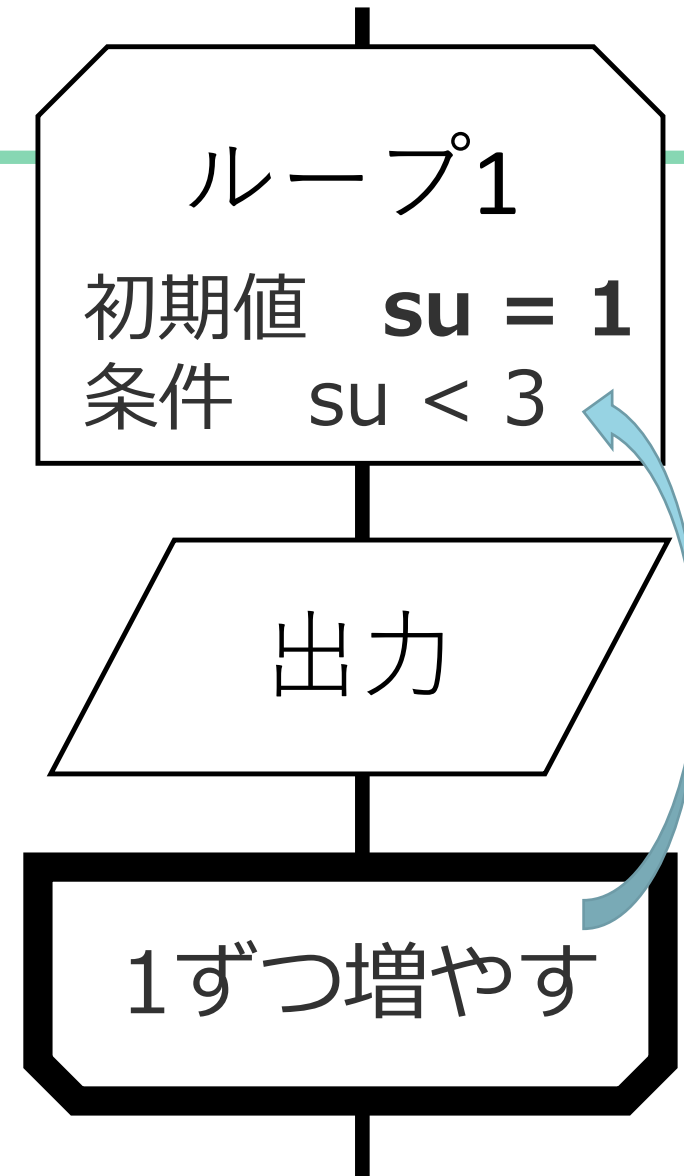
Print output

0
1
2

for 繰り返し

```
for su in range(1,3):  
    print (su)
```

繰り返したい処理



for 繰り返し

```
1 for su in range(1,3):  
2     print(su)
```

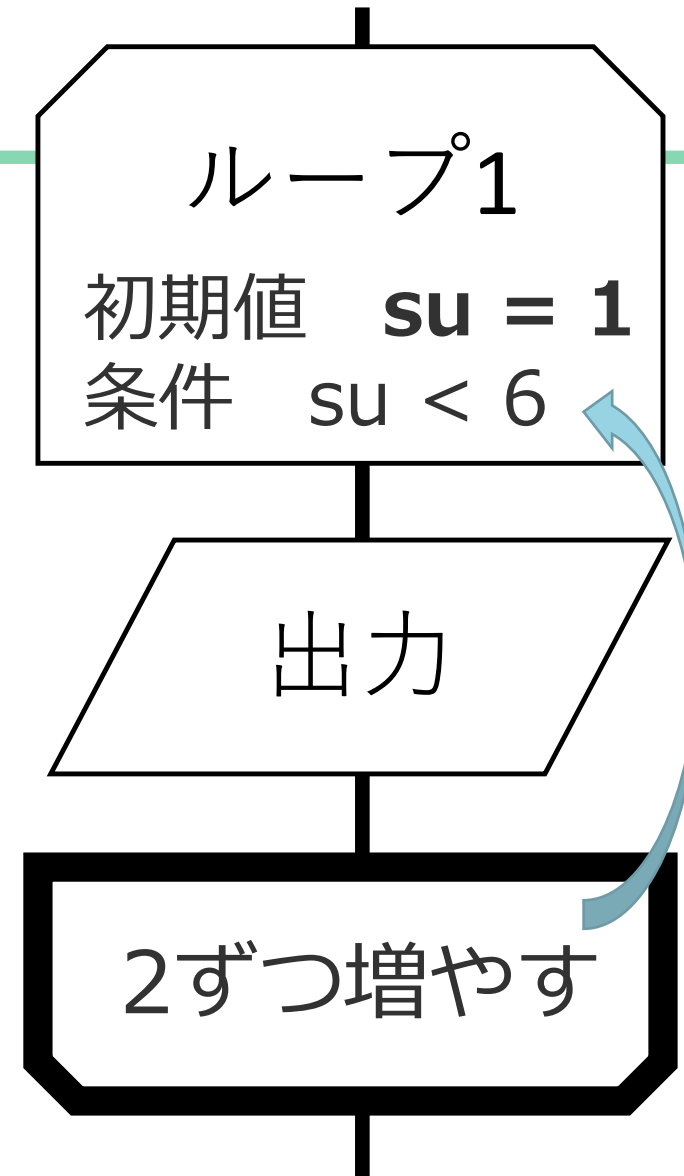
Print (

1
2

for 繰り返し

```
for su in range(1,6,2) :  
    print (su)
```

繰り返したい処理



for 繰り返し

```
1 for su in range(1, 6, 2):  
2     print(su)
```

Print output

```
1  
3  
5
```

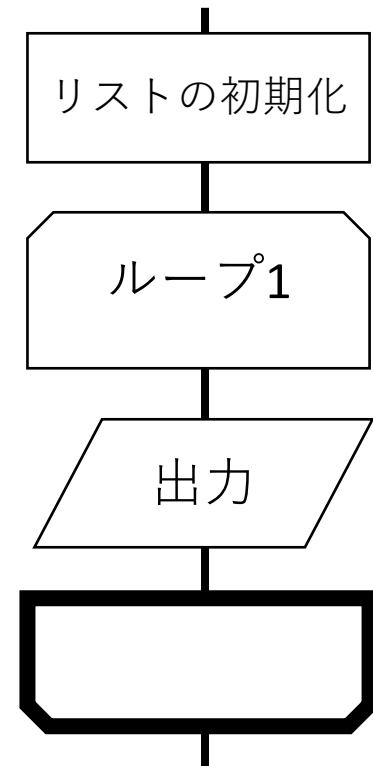

for 繰り返し

```
coins = [1, 5, 10, 50, 100, 500]
```

```
for yen in coins:
```

```
    print(yen, '円')
```

繰り返したい処理



for 繰り返し

```
1 coins = [1, 5, 10, 50, 100, 500]
2 for yen in coins:
3     print(yen, ' 円 ')
```

1	円
5	円

確認問題 7

2から10までの偶数だけを出力する

ただし、if (条件) を使用せずに 作成すること

確認問題 7

2から11未満までの間2ずつ増やす

```
for su in range(2, 11, 2):
```

```
    print(su)
```

確認問題 8

1から10までの偶数だけを出力する

ただし、if (条件) を使用して 1～10までの

数値の内 偶数のときだけ出力する プログラム

を作成すること

確認問題 8

1から11未満までの間1ずつ増やす

```
for su in range(1, 11, 1):
```

```
    if (su % 2) == 0:
```

```
        print(su)
```

確認問題 9

1から99までの整数の内、7で割り切れる数
を表示する

ただし、利用できる変数は `su` のみとし、

繰り返しは `while` を使用して 作成すること

確認問題 9

```
su = 1
```

```
while su <= 99:
```

```
    if su % 7 == 0:
```

```
        print(su)
```

```
    su = su + 1
```