Python部品カード

次

01	文字や数値を表示する
02	四則演算と変数への代入
03	キーボードから数値や文字の入力
04	条件を判断して、命令を実行
05	条件を判断して、〇と×で違う命令を実行
06	複数の条件を判断して違う命令を実行
07	0からnまで繰り返す
08	指定回数繰り返す:range()
09	リストの処理(取り出し、入れ替え)
10	二つの変数の内容を入れ替える
11	二重の繰り返し
12	二重の繰り返し(内の繰り返しの開始を変更)
13	文字列の構造と操作
14	数字と数値の変換/文字列の構造と操作
15	条件の成り立つ間繰り返す
16	乱数を使用する
17	二次元リスト(配列)を使用する
18	関数の定義と利用
19	リストの初期化と追加
20	再帰呼び出し
21	文字コード
22	論理演算(and /or)
23	if –elif -else
24	ループの処理状態判定(フラグ利用)
25	

Pythonプログラム機能部品カード　No.01

文字や数値を表示する

画面に文字や数値を表示します。

部品
01

具体例

画面に Hello と表示する。

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

画面に 3 + 7の計算結果を表示する。

```
print(3+7)
```

変数Xに数を入れた後に
画面に 変数Xの内容を表示する。

```
x = 25
print(x)
```

裏面

Pythonプログラム機能部品カード　No.01

文字や数値を表示する

図式例

表示する(Hello)

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

表示する(3+7)

```
print(3+7)
```

四則演算と変数への代入

足し算、引き算、掛け算、割り算の基本的な使い方と、数値や計算結果の変数への代入方法です。

部品 02

具体例

変数iの内容に1を足した値

```
i + 1
```

変数xと変数iをかけた値

```
x * i
```

変数xの内容から2を引いた値

```
x - 2
```

変数aを10で割った値

```
a / 10
```

変数 a, 変数b, 変数cの内容を足す

```
a + b + c
```

変数aの内容を0にする。

```
a = 0
```

変数Xの内容を変数Iに1を足したものにする。

```
x = a + 1
```

裏面

Pythonプログラム機能部品カード　No.02

四則演算と変数への代入

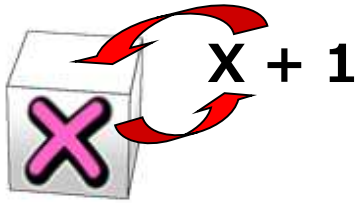
図式例

```
a = 0
```

```
a = 0
```

```
x = x + 1
```

```
x = x + 1
```



x = 1

x = x +1
(x ← x + 1)

xと名前をつけた箱(変数)に1を入れる

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。

キーボードから数値や文字の入力

キーボードから数値や文字を入力して変数に入れます。

部品
03

具体例

キーボードから入力した文字を変数xに入れる。

```
x = input()
```

キーボードから入力した数字を整数として変数aに入れる。

```
a = int(input())
```

Input()は文字として入力します。

例えば100とキーを打っても
100という数字として入力されます。

そこで、int( )を使用して数字→数値に
変換して計算などに使用します。

裏面

Pythonプログラム機能部品カード　No.03

キーボードから数値や文字の入力

図式例

x = (入力)

```
x = input()
```

a = (数値入力)

```
a = int(input())
```

Pythonプログラム機能部品カード　No.04

条件を判断して、命令を実行

条件を判断して、条件が成り立つ時に命令を実行します。

具体例

部品
04

変数aの内容が70より大きければ、合格(Goukaku)と表示する。

```
if a > 70:
    print("Goukaku")
```

変数aの内容が、変数bの内容より小さければ、変数xに
変数bの内容を入れる。

```
if a < b:
    x = b
```

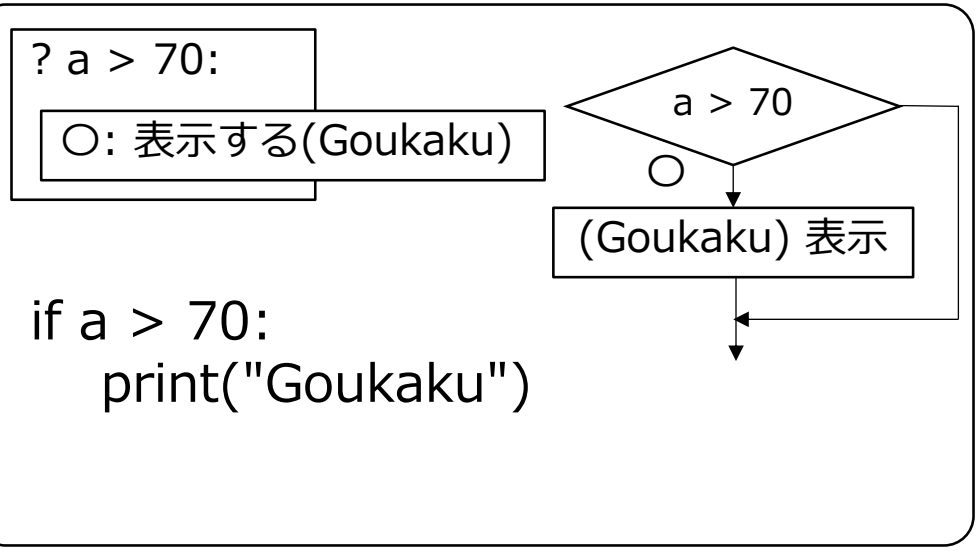
x == y	x と y が等しい
x != y	x と y が等しくない
x > y	x は y よりも大きい
x < y	x は y よりも小さい
x >= y	x は y と等しいか大きい
x <= y	x は y と等しいか小さい

裏面

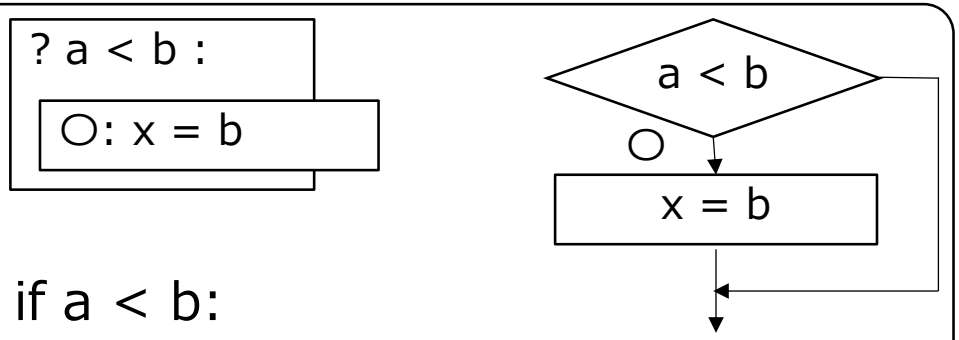
Pythonプログラム機能部品カード　No.03

条件を判断して、命令を実行

図式例




```
if a > 70:
    print("Goukaku")
```




```
if a < b:
    x = b
```

Pythonプログラム機能部品カード　No.05

条件を判断して、〇と×で違う命令を実行

条件を判断して、条件が成り立つ時と成り立たない時に別々の命令を実行します。

部品 05

具体例

変数xの内容が19より大きいか判断して。大きければ成人(Seijin)と表示し、そうでなければ、未成年(Miseinen)と表示する。

```
if x > 19:
    print("Seijin")
else:
    print("Miseinen")
```

変数aの内容が変数bの内容より大きいか判断して。大きければ変数xに変数aの内容を入れ、そうでなければ、変数xに変数bの内容を入れる

```
if a > b:
    x = a
else:
    x = b
```

裏面

Pythonプログラム機能部品カード　No.05

条件を判断して、〇と×で違う命令を実行

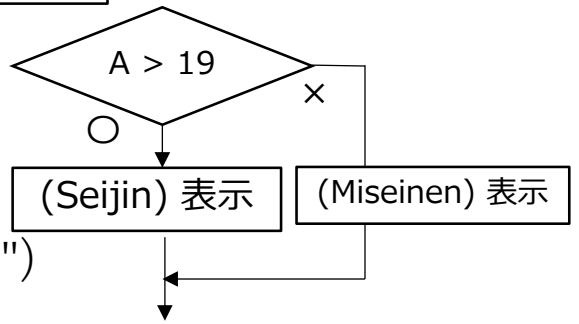
図式例

? x > 19:

〇: 表示する(Seijin)

×: 表示する(Miseinen)

```
if x > 19:
    print("Seijin")
else:
    print("Miseinen")
```

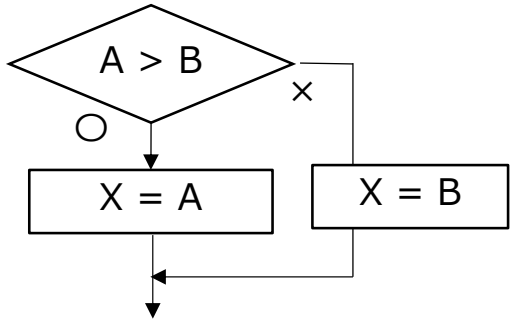


? a > b:

〇: x = a

×: x = b

```
if a > b:
    x = a
else:
    x = b
```



Pythonプログラム機能部品カード　No.06

複数の条件を判断して違う命令を実行

複数の条件を判断して、違う命令を実行します。
No.04やNo.05で、実行する命令として、No.04やNo.05自体を使用します。

部品 06

具体例

```
if a < 6:
    print("Youji")
else:
    [          ]
```

←

```
if a < 12:
    print("Kodomo")
else:
    print("Otona")
```

条件を判断する二つの部品を組み合わせます。

↓

```
if a < 6:
    print("Youji")
else:
    if a < 12:
        print("Kodomo")
    else:
        print("Otona")
```

まず、変数aの内容が6未満であるか判断して、未満であれば、幼児(Youji)と表示します。

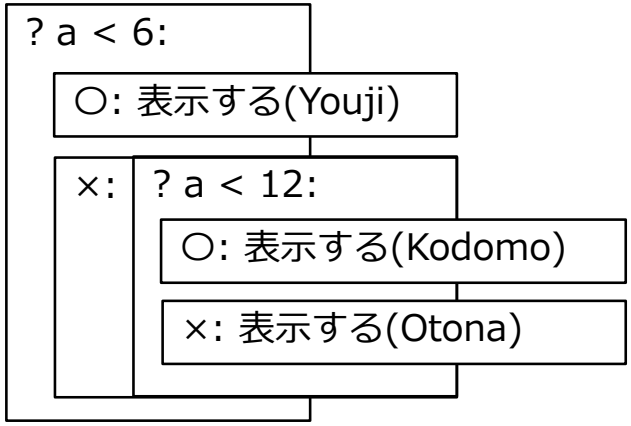
もし、そうでなければ、今度は、変数Aの内容が12未満であるか判断して、未満であれば、子供(Kodomo)と表示します。そうでなければ、大人(Otona)と表示します。

裏面

Pythonプログラム機能部品カード　No.05

複数の条件を判断して違う命令を実行

図式例




```
if a < 6:
    print("Youji")
else:
    if a < 12:
        print("Kodomo")
    else:
        print("Otona")
```

字下げしていることで、elseの中で実行する命令となっている。

Pythonプログラム機能部品カード　No.07

0からnまで繰り返す

ある変数の内容を0からnまでカウントアップしながら、繰り返して処理します。

部品
07

具体例

変数iの内容を0, 1, 2, 3 …と10までカウントアップして、処理を繰り返しています。

```
for i in range(11):
    print(i)
```

<- 変数iの内容を0,1,2,3, ・・,10まで変えて繰り返しで実行します。

繰り返しの中で実行する命令としてprint(i)で、iの内容を表示しています。

range(n)の意味
　0から始まってn以下の数まで、1づつカウントアップしたデータを作ります。

0,1,2,3,4,5,6,7,8,9,10

裏面

Pythonプログラム機能部品カード　No.07

0からnまで繰り返す

図式例

```
for i in range(11):
```

繰り返しで実行する命令の集まり(変数iの利用含む)

i = [1,2,…,10]
繰り返す

繰り返しで実行する命令の集まり(変iの利用含む)

補足

```
for i in range(11):
    print(i)
    print(i*2)
    print(i*3)
```

↑ 字下げしていることで、for命令の中で3個のprint命令が実行される。

Pythonプログラム機能部品カード　No.08

指定回数繰り返す:range()

部品
08

ある変数の内容を初めの数から、終わりの数まで、
指定した数だけでカウントアップしながら、繰り返
して処理します。

具体例

変数iの内容を1から10未満の数まで2づつカウントアップして、
処理を繰り返しています。

```
for i in range(1,10,2):
    print(i)
```

<- 変数iの内容を
1から10未満まで2づつ変えて
繰り返しで実行します。

range(1,10,2):[1,3,5,7,9]

変数iの内容を10から100未満の数まで10づつカウントアップして、
処理を繰り返しています。

```
for i in range(10,100,10):
    print(i)
```

<- 変数iの内容を
10から100未満まで10づつ
変えて繰り返しで実行します。

range(10,100,10):[10,20,30,・・,80,90]

裏面

Pythonプログラム機能部品カード　No.08

指定回数繰り返す:range()

Range()のパラメータ

range(終了値)

range(5):[0,1,2,3,4]

range(開始値, 終了値, 増分)

range(1,10,2):[1,3,5,7,9]

range(10,100,10):[10,20,30,・・,80,90]

Pythonプログラム機能部品カード　No.09

リストの処理(取り出し、入れ替え)

リストの中の個々の要素の取り出し、入れ替え、
追加、クリアをします。

部品
09

具体例

d[3]

リストdの4番目の内容です。

d[i]

リストdの(iの変数の内容) の内容です。
例えばiに4が入っていたら5番目の内容です。

d[2] = 2

リストdの3番目の内容を2にします。

d[i] = x

リストdの(iの変数の内容) の内容を変数Xの内容にします。
例えばiに4、xに100か入っていたら5番目の内容を100にです。

裏面

Pythonプログラム機能部品カード　No.09

リストの処理(取り出し、入れ替え)

図式例

```
x = d[3]
```

x = d[3]

```
? d[i] < 10 :
    ○: (処理)
```

```
if d[i] < 10:
    (処理)
```

```
d[i] = 2
```

d[i] = 2

Pythonプログラム機能部品カード　No.10

二つの変数の内容を入れ替える

二つの変数の内容を入れ替えます。入れ替える時、片一方の変数の内容を一時的に他の変数に入れるのがポイントです。

部品
10

具体例

```
t = a
a = b
b = t
```

変数aと変数bの内容を入れ替えます。(変数aにbの内容を入れる前に、変数aの内容をtに入れて退避させています)

リストd[0]の箱の内容とd[i]の内容を入れ替えています。

```
t = d[0]
d[0] = d[i]
D[i] = d[0]
```

裏面

Pythonプログラム機能部品カード　No.010

二つの変数の内容を入れ替える

図式例

A <-> b

```
t = a
a = b
b = t
```

又は

t = a
a = b
b = t

d[0] <-> d[i]

```
t = d[0]
d[0] = d[i]
D[i] = d[0]
```

又は

t = d[0]
d[0] = d[i]
d[i] = t

補足

Pythonの場合は、下記のようにしても入れ替えできます。

```
a, b = b, a
```

```
d[0], d[i] = d[i], d[0]
```

Pythonプログラム機能部品カード　No.11

二重の繰り返し

ある変数の内容をカウントアップする繰り返しを二つ組み合わせて使用します。

部品 11

具体例

```
for i in range(5):
    繰り返しで実行する命令の集まり(変数iの利用含む)
    for j in range(5):
        print(d[i] * d[j])
```

変数iでカウントアップ　　変数Iでカウントアップ

繰り返しの二つの部品を組み合わせます。

```
for i in range(5):
    for j in range(5):
        print(d[i] * d[j])
```

変数iの内容が0,1,・・4とカウントアップして下記の変数jを使った繰り返しを5回繰り返します。

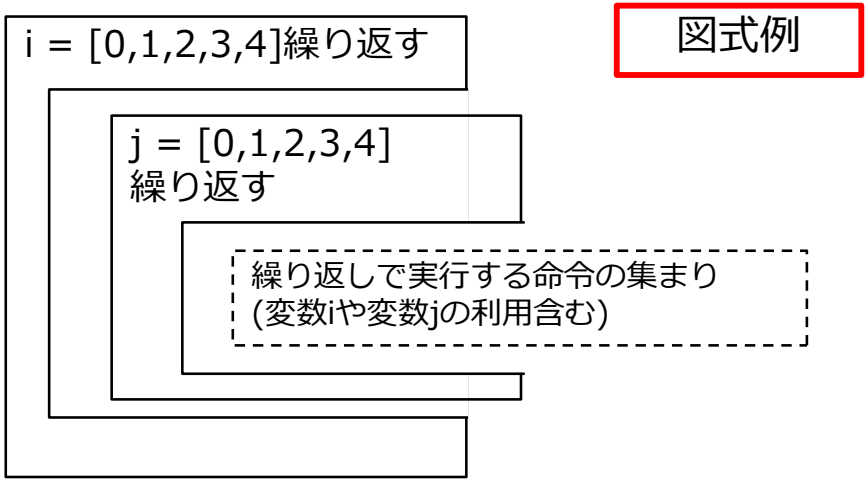
変数jの内容が0,1,・・4とカウントアップして表示します。

裏面

Pythonプログラム機能部品カード　No.11

二重の繰り返し

図式例




```
for i in range(5):
    for j in range(5):
        繰り返しで実行する命令の集まり
        (変数iや変数jの利用含む)
```

Pythonプログラム機能部品カード　No.12

二重の繰り返し(内の繰り返しの開始を変更)

ある変数の内容をカウントアップする繰り返しを二つ組み合わせて使用します。但し、内側のカウントアップする数の初めを外側の繰り返しの変数の内容にしています。

部品 12

具体例

```
for i in range(5):
```

繰り返しで実行する命令の集まり(変数iの利用含む)

```
    for j in range(i,5):
        print(d[i] * d[j])
```

変数iでカウントアップ

変数iでカウントアップ
初めにiを設定(処理内はiから)

繰り返しの二つの部品を組み合わせます。

```
for i in range(5):
    for j in range(i,5):
        print(j)
```

変数iの内容が0,1, 2, 3, 4 とカウントアップして下記の変数Jを使った繰り返しを5回繰り返します。

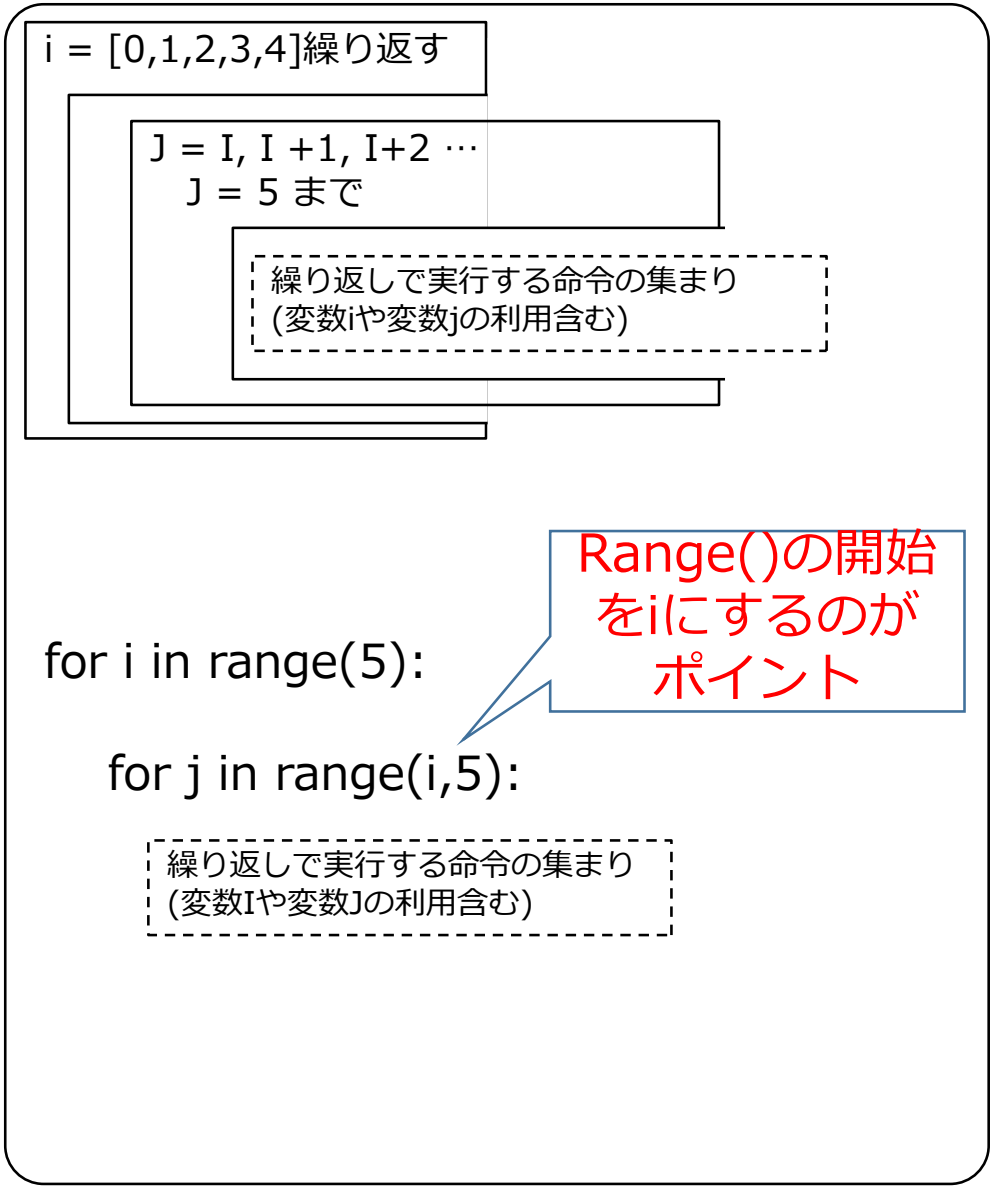
変数jの内容が、個々の繰り返しが始まった時のiの内容から始めるので
0, 1, 2, 3, 4,
1, 2, 3. 4
2, 3, 4
3, 4
4 とカウントアップして表示します。

裏面

Pythonプログラム機能部品カード　No.12

二重の繰り返し(中の繰り返しの開始を変更)

図式例




```
for i in range(5):
    for j in range(i,5):
```

Range()の開始をiにするのがポイント

繰り返しで実行する命令の集まり
(変数Iや変数Jの利用含む)

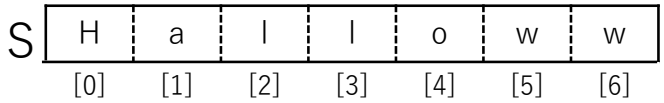
文字列の構造と操作

Pythonの文字列はリストと同様に添え字で利用することができます。

部品 13

具体例

S = “Hallow”



S[1]　文字列の2番目の文字です　具体例では　a

S[i]　文字列Sの(iの変数の内容) の内容です。
例えばiに4が入っていたら5番目の内容です。

次

Pythonプログラム機能部品カード　No.13

文字列の構造と操作

具体例

文字列の大きさ　len(文字列)

len(S)　Sの文字列の大きさ(長さ)
　S = “Hallow”の場合、len(S)は6

文字列の結合　文字列 + 文字列

S = “Hallow”+”!”
W = “world”
S2 = S + W の時

S2は “Hallow!world”

補足

S[i] = “A”　エラーになる

Pythonの文字列はリストと異なり固定値であるため、変更はできない。

数字と数値の変換/文字列の構造と操作

Pythonの場合は文字と数値は区別されています。そのため
相互に操作する場合は、変換が必要になります。

部品
14

具体例

キーボードから入力した数字を整数として変数aに
入れる。

```
a = int(input())
```

int( )を使用して数字→数値に変換して
計算などに使用します。

数値に数字に変換して文字に追加する

```
a = 10
b = “No”
c = b + str(a)
```

数値をそこで、int( )を使用して数字→
数値に変換して計算などに使用します。

次

Pythonプログラム機能部品カード　No.14

文字列の構造と操作

具体例

文字(数値) ->整数に変換 int()

S = “5”
a = int(S)の場合、a は5という数値

文字(数値) ->小数に変換 float()

S = “3.14”
a = int(S〉b=float(S)場合、
a は3という数値, bは3.14という数値

数値 -> 文字(数字) に変換　str()

a = 10
b = 10.11
c = str(a) + str(b) の場合
c は1010.11sという文字列

Pythonプログラム機能部品カード　No.15

条件の成り立つ間繰り返す

指定した条件が成り立つ間、処理を繰り返す。

部品
15

具体例

変数iの内容を0, 1, 2, 3 …と10までカウントアップして、11未満の間、処理を続けます

```
i = 0
while i < 11:
    print(i)
    i = i + 1:
```

変数iの内容を
0,1,2,3, ・・,と変えて、iが11未満の時まで処理を利繰り返しています
whileに入るまえに、条件判断で使用するiを0にしています。
またwhileの中でiの値を変更しています

次

Pythonプログラム機能部品カード　No.15

条件の成り立つ間繰り返す

図式例

```
while i < 11
```

i < 11の間
繰り返す

繰り返しで実行する
命令の集まり(変数iの
利用含む)

補足

```
while i < 11:
    print(i)
    sum = sum + i
    i = i + 1
```

↑ 字下げしていることで、while命令の中で3個の命令が実行される。

Pythonプログラム機能部品カード　No.16

乱数を使用する

整数や小数点の乱数を使用する

部品
16

具体例

randomモジュールを取り込む

```
import random
```

１～10までの整数の乱数を変数aに代入する。

```
a = random.randint(1, 10)
```

0.0～1.0までの小数点の乱数を変数bに代入する。

```
a = random.random()
```

次

Pythonプログラム機能部品カード　No.16

文字や数値を表示する

図式例

乱数(1～6の整数)

```
random.randint(1, 6)
```

乱数(0～1の小数点)

```
random.random()
```

補足

乱数の関数を使用する場合はrandomモジュールを
金須らず取り込む

```
import random
```

Pythonプログラム機能部品カード　No.17

二次元リスト(配列)を使用する

Pythonのリストの要素としてリストを使用すると
二次元リストとして定義、操作することかできかる

部品
17

具体例

複数のX,Y座標を定義するリスト

```
Pos = [[0 ,0], [100,100], [150,100]]
同じ内容で改行したもの
Pos = [[0 ,0],
       [100,100],
       [150,100]]

len(Pos)  Pos要素の数 3
len(Pos[1]) 1番目の要素の内容の要素の数 2

Pos[0]   0番目の要素   [0, 0]
Pos[1][0] 1番目の要素の0番目の要素   100

print(Pos) Posの内容の表示
   [[0 ,0], [100,100], [150,100]]

print(Pos[1]) Pos[1]の内容の表示
[100,100]
```

次

Pythonプログラム機能部品カード　No.17

二次元リスト(配列)を使用する

具体例

複数の数値と文字列を組み合わせたリスト

```
P = [[150 ,”パン”], [200,”牛乳”], [250,”チーズ”]]
同じ内容で改行したもの
P  = [[150 ,”パン”],
      [200,”牛乳”],
      [250,”チーズ”]]

len(P)  P要素の数 3
len(P[1]) 1番目の要素の内容の要素の数 2

P[0]   0番目の要素    [150 ,”パン”]
P[1][0] 1番目の要素の0番目の要素   200

print(P) Pの内容の表示
 [[150 ,”パン”], [200,”牛乳”], [250,”チーズ”]]

print(P[1]) P[1]の内容の表示
[200,”牛乳”]
```

関数の定義と利用

Pythonではdefブロックで新しい関数を定義して、
それを利用することができる。

部品
18

具体例

二つの数を加える関数の定義と利用

```
def addvalue( a, b)
    c = a + b
    return   c

print(addvalue(10,20))
30
```

関数の定義

関数の利用と
実行結果

二乗した数を求める関数の定義と利用

```
def svalue( a)
    b = a * a
    return b

print(addvalue(10))
100
```

関数の定義

関数の利用と
実行結果

次

Pythonプログラム機能部品カード　No.18

文字や数値を表示する

図式例

関数を使用する場合

d = 関数: addvalue(10,　20)

```
d = addvalue(10,20)
```

関数を定義する

addvalue(a, b)

```
def addvalue( a, b):
```

関数の戻り値を指定する

戻る: c

```
    return c
```

重要

関数の定義は、使用する前で行うこと

リストの初期化と追加

リストを処理する場合、その範囲を超える要素を利用しようとするとpythonでは、あらかじめ十分な要素数の両院を確保したり、リストに追加する処理が必要になります。

部品 19

具体例

1,2,3,4,5の数字が入ったリストを定義する。

```
S = [1, 2, 3, 4, 5]
```

すべて内容が0の10個の要素のリストを代入する。

```
S = [0] * 10
```

参考　Sの内容の確認

```
print(S)
[0,0,0,0,0,0,0,0,0,0]
```

要素の入っていないリストを定義する

```
S = []
```

次

Pythonプログラム機能部品カード　No.19

リストの初期化と追加

具体例

1,2,3,4,5の数字が入ったリストに6の要素を追加する。

```
S = [1, 2, 3, 4, 5]
S.append(6)
```

参考　Sの内容の確認

```
print(S)
[1, 2, 3, 4, 5, 6]
```

図式例

リストPに6の追加

```
P.append(6)
```

再帰呼び出し

漸化式(数学Bで学習)のように、ある一定の規則を持つ数列において、その数列が各項とも、前の項の関係で表現したものであり、再帰呼び出しは、漸化式をプログラムで直接実現したものである。

部品 20

具体例

例えば5!(階乗)は
　5 * 4 * 3 * 2* 1 の値を持ち
f(1) =1, f(n) = n * f(n-1)の漸化式で表現できる

```
def f(n):
    if n ==1:
        v = 1
    else:
        v = n * f(n-1)
    return v
```

確認

```
print(f(5))
120
```

次

Pythonプログラム機能部品カード　No.20

再帰呼び出し

動作イメージ

```
def f(n):
    if n ==1:
        v = 1
    else:
        v = n * f(n-1)
    return v
```

f(5)の値

```
v = 5*f(4) ← 5 * 4 * 3 *2 * 1
        = 4*f(3) ← 4 * 3 * 2 * 1
              = 3*f(2) ← 3 * 2 * 1
                    = 2 * f(1) ← 2 * 1
                          = 1
```

文字コード

コンピュータ内部で文字を扱う場合は、各文字に対応して、バイト単位の数値が割り当てられて処理される。その文字と数値の対応を文字コードと呼んでいる。
Pythonでは、ASCII文字コードとして文字から文字コードに変換する場合は、ord()関数、文字コードから文字に変換する場合は、chr()関数を使用する

部品
21

具体例

Sに入っている文字のコードを表示する ord関数

```
c = “A”
v = ord(s)
print(v)
65
```

66に対応する文字を表示する。

```
v = 66
c = chr(v)
print(c)
B
```

次

Pythonプログラム機能部品カード　No.21

文字コード

ASCII文字コード表

下位4ビット＼上位3ビット	0	1	2	3	4	5	6	7
0				0	@	P	`	p
1			!	1	A	Q	a	q
2			"	2	B	R	b	r
3			#	3	C	S	c	s
4			$	4	D	T	d	t
5			%	5	E	U	e	u
6			&	6	F	V	f	v
7			'	7	G	W	g	w
8			(	8	H	X	h	x
9			)	9	I	Y	i	y
10			*	:	J	Z	j	z
11			+	;	K	[	k	{
12			,	<	L	¥	l	|
13			-	=	M	]	m	}
14			.	>	N	^	n	~
15			/	?	O	_	o	

例

文字Aの場合

ビット表現	1	0	0	0	0	0	1

上位3ビット　下位4ビット

16進数表現　41
10進数表現　65　(16*4 +1)

論理演算(and /or)

論理演算子は、a かつ b や　a または b などの真偽の評価するときの演算子です。if, while などで使用します。

部品
22

具体例

aが0未満かつbが100未満の間、繰り返します。

```
while a < 0 and b <100:
    (実行命令)
```

a==0 または b==1の時、if文の処理を実行します。

```
if a == 0 or b ==1:
    (実行命令)
```

次

Pythonプログラム機能部品カード　No.22

論理演算(and /or)

and / orの評価

a and b (かつ)

a	b	a and b
偽	偽	偽
偽	真	偽
真	偽	偽
真	真	真

a or b (または)

a	b	a and b
偽	偽	偽
偽	真	偽
真	偽	偽
真	真	真

Pythonプログラム機能部品カード　No.23

if –elif -else

複数の条件連続して判断する場合、if文の後にelifを続けることで、指定することができます。

部品
23

具体例

aの値が、
70より大きければ、成績Aと表示
30より大きければ、成績Bと表示
それ以外の時は、成績Cと表示

```
if a > 70:
    print(“成績A”)
elif a > 30:
    print(“成績B”)
else:
    print(“成績C”)
```

補足

elifは複数回使うことができます。

```
if (条件):
elif (条件):
elif (条件):
 ・・・
else:
```

次

Pythonプログラム機能部品カード　No.23

if –elif -else

```
if a > 70:
    print(“成績A”)
elif a > 30:
    print(“成績B”)
else:
    print(“成績C”)
```

⇔

```
if a > 70:
    print(“成績A”)
else:
    if a > 30:
        print(“成績B”)
    else:
    print(“成績C”)
```

if –elif は、上手のようにif-elseの中にifを入れたことになります。

ループの処理状態判定(フラグ利用)

部品
24

forやwhileでは、プログラムの問題とは一見関係ない数値を設定する場合があります。これらは、カウンターというより、状態を把握するためのフラグ(旗)の追加谷なります。

具体例

i = 999と処理では使用しない数値を設定して、正しく状態になるとこの値を変更してwhileを抜け出します。

```
i = 999
while i < 999:
    if xxxx:
     i = 3
```

変数iの内容を999に設定してwhileに入ります。
whileの中で処理を繰り返し、ある一定の条件になったら、iに999以外の数値を設定することでwhileを抜け出します。

forの前にfにforの中では入らない数値(この例では-1)を設定して、forの後で、forの中で何らかの処理が行われたか判断します。

```
f = -1
for I in xxxx:
    if xxxxx:
        f = 3
if f == -1:
    xxxx
else:
    xxxx
```

forの中では、ある条件の時fの値を変更します。
forの後でfの値を判断することで、forの中で何らかの処理が行われたかどんか判断できます。

次

Pythonプログラム機能部品カード　No.15

ループの処理状態判定(フラグ利用)

補足

```
i = 999
while i < 999:
      i = 3

f = -1
for I in xxxx:
    if xxxxx:
      f = 3
```

フラグとして使用する変数に事前に設定する値について大きく二つの方法があります。
① 処理で設定される数より十分に大きな数の場合
② 処理の中で設定される数が正の数の場合には、負の数を設定する場合。

注意

```
i = 999
while i < 999:
    if xxxx:
     i = 3
```

Whileの終了条件の確認:

フラグとして変数をWhileで使用する場合は、Whileの中で確実に終了するように変数の値が設定するか確認する必要があります。
ここでエラーがあると無限ループになり止まらないプログラムになります。