

The Kyoto College of
Graduate Studies
for Informatics

kcg.edu

コンピュータプログラミング概論

2021年秋期第4回 eラーニング資料

安 平勲

h_an@kcg.ac.jp

制御構造：分岐と反復

- コンピュータは命令を上から下へ**順次**処理しようとする

自分のBMIを計算してみよう

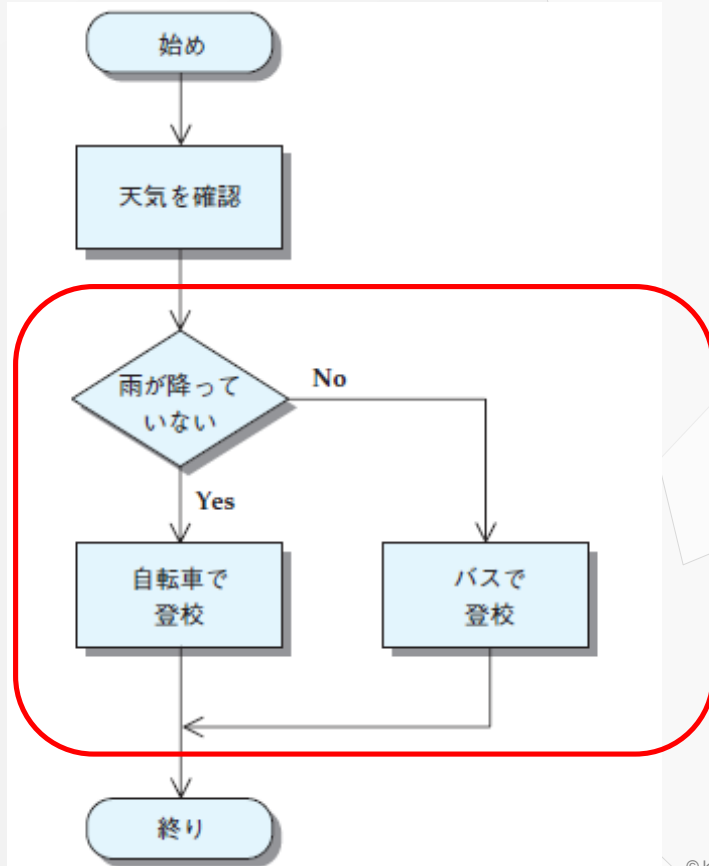
```
In [9]: ▶ # 自分のBMIを計算
          tall = 1.70           # 身長
          weight = 68           # 体重
          bmi = weight / tall / tall # 計算式
          print(bmi)           # 25以上は肥満

23.529411764705884
```

- 順次の他に『条件**分岐**』と『**反復**』の基本処理がある
- **順次・分岐・反復**が構造化プログラミングの考え

条件分岐

家を出るときに天気を確認し、
雨が降っていないければ、
自転車で行く、
しかし、雨が降っていれば、
バスで大学に行く



Pythonの条件分岐： **if** , **else**, **elif**

■ 単純分岐

if 条件式：
→ 処理
字下げ

➤ **もし**条件式の値が**True**（真）
なら**処理**を**実行**し、
False（偽）なら**実行しない**

- Pythonは処理行の先頭に「**字下げindent**」を**必ず**入れる！
 - Spyder/Jupyterの場合、改行すればインデントが自動的に入る。
半角空白文字や**TAB**を使ってインデントを自分で入れてもいい

単純分岐

- 変数varが10かどうか判別するプログラム

```
var = 10
```


```
if var == 10 :
```

```
    print("varは10です")
```

```
print("終了します")
```

- ① 「var=20」に変更してもう一度試そう
- ② 「print("終了します")」を字下げして試そう

- if文の次の行を字下げしている
- 行頭の空白の数（インデント数）が意味を持つ

 インデント数は半角空白**4**つが推奨されている

Pythonの条件分岐： **if** , **else**, **elif**

■ 二分岐

if 条件式：

処理1 # 条件式が**True**の場合、実行

else：

処理2 # 条件式が**False**の場合、実行

- **もし**条件式の値が**True**（真）なら**処理1**を**実行**し、**それ以外**（**False**）なら**処理2**を**実行**する

二分岐

- varが10かどうか判別する

```
var = 20
```

```
if var == 10:
```

```
    print("varは10です")
```

```
else:
```

```
    print("varは10ではありません")
```

```
print("終了します")
```

インデントindent

- Pythonでは、if ・ else ・ elifの後の分岐処理ブロックは字下げ（インデント）しないとエラー!!

※ifやelseの処理範囲をインデント（+コロン`:`とセット）で判断

```
var = 20
```

```
if var == 10:
```

```
    print("varは10です")
```

```
    print("varが10の場合の処理")
```

```
else:
```

```
    print("varは10ではありません")
```

```
        print("varが10でない場合の処理")
```

```
print("終了します")
```


■ 多分岐

if 条件式1 :

条件式1が**True**の場合, 実行

処理1

elif 条件式2 :

条件式1が**False**, かつ条件式 2 が**True**の場合, 実行

処理2

else :

条件式1が**False**, かつ条件式 2 が**False**の場合, 実行

処理3

多分岐

- varが10か20かそれ以外か判別する

```
var = 20
```

```
if var == 10:
```

```
    print("varは10です")
```

```
elif var == 20:
```

```
    print("varは20です")
```

```
else:
```

```
    print("varは10でも20でもありません")
```

```
print("終了します")
```

ブール型演算子 ; and/or/not

```
In [1]: ▶ a = 10 ; b = 10 ; c = 3
```

```
In [2]: ▶ ( a == b ) and ( a == c )
```

```
Out[2]: False
```

```
In [3]: ▶ ( a == b ) or ( a == c )
```

```
Out[3]: True
```

```
In [4]: ▶ not ( a > b )
```

```
Out[4]: True
```

条件式の例

ブール型演算子

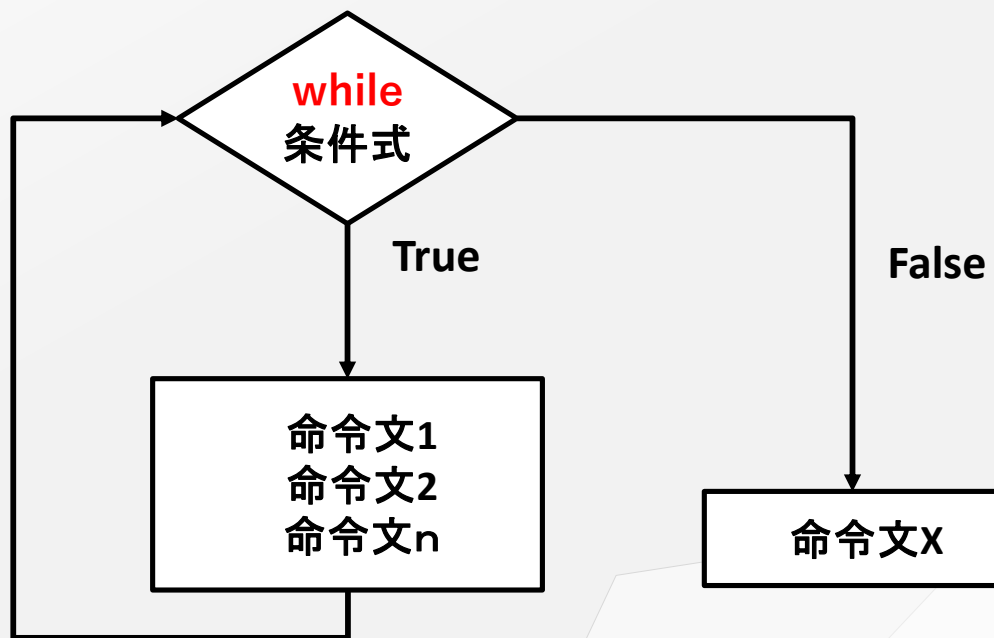
- “and” – すべての論理式がTrueの時True。論理積
- “or” – 少なくとも一つがTrueの時True。論理和
- “not” – 否定
 $\text{not True} == \text{False}$
 $\text{not False} == \text{True}$

Pythonの反復処理： while


while 条件式：
条件式が**True**の間, 命令文,1, 2, ...nを実行
命令文1
命令文2
:
命令文n
命令文X # インデント外の命令文Xはwhileの条件式がFalseの時実行

Pythonの反復： while

- Whileの反復ループをフローチャートで書くと・・・



Pythonの反復： while



```
count = 1
while count <= 10 :
    print(count)
    count += 1
print('while 終了')
```

```
1
2
3
4
5
6
7
8
9
10
while 終了
```

➤ 『count』が10以下のとき反復（ループ）処理を実行



✓ 『+=』は複合代入演算子（次頁）

複合代入演算子

- よく使うソースコードのタイプ数を減らせる

代入演算子	例	等価
<code>+=</code>	<code>a += 1</code>	<code>a = a + 1</code>
<code>-=</code>	<code>a -= 2</code>	<code>a = a - 2</code>
<code>*=</code>	<code>a *= 3</code>	<code>a = a * 3</code>
<code>/=</code>	<code>a /= 10</code>	<code>a = a / 10</code>
<code>//=</code>	<code>a //= 10</code>	<code>a = a // 10</code>
<code>%=</code>	<code>a %= 10</code>	<code>a = a % 10</code>
<code>**=</code>	<code>a **= 10</code>	<code>a = a ** 10</code>

While ; 永久ループとbreak

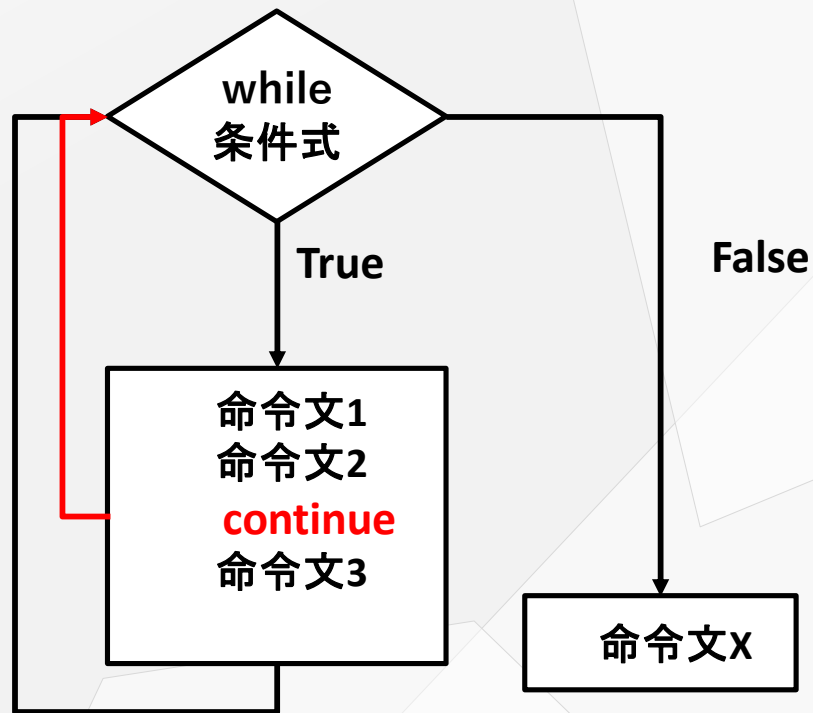
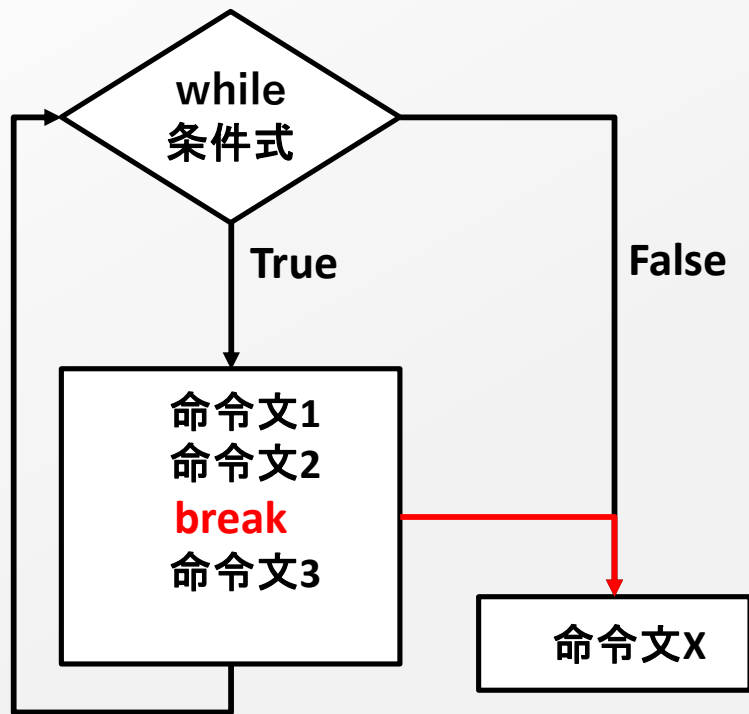
```
▶ count = 10
while True :
    print(count)
    count -= 1
    if count < 1:
        break
print('while 終了')
```

10
9
8
7
6
5
4
3
2
1

while 終了

- 条件式は常に『True』なので、ループを無限に実行
- 『break』でwhileループを終了

While ; breakとcontinue



While ; continue

■ 10までの奇数だけを出力

```
▶ count = 0
while count < 10:
    count += 1
    if count % 2 == 0:
        continue
    print(count)
print('while 終了')
```

1
3
5
7
9
while 終了

- 偶数(余りが0)の場合, 次のprintは実行しない (skip)
* : whileループは継続

Pythonの反復： for in

for 変数 **in** range (n) :

 命令文1

 命令文2

 命令文3

命令文1,2,3を n 回繰り返す

※変数には0,1, 2, ... $n-1$ が順に設定

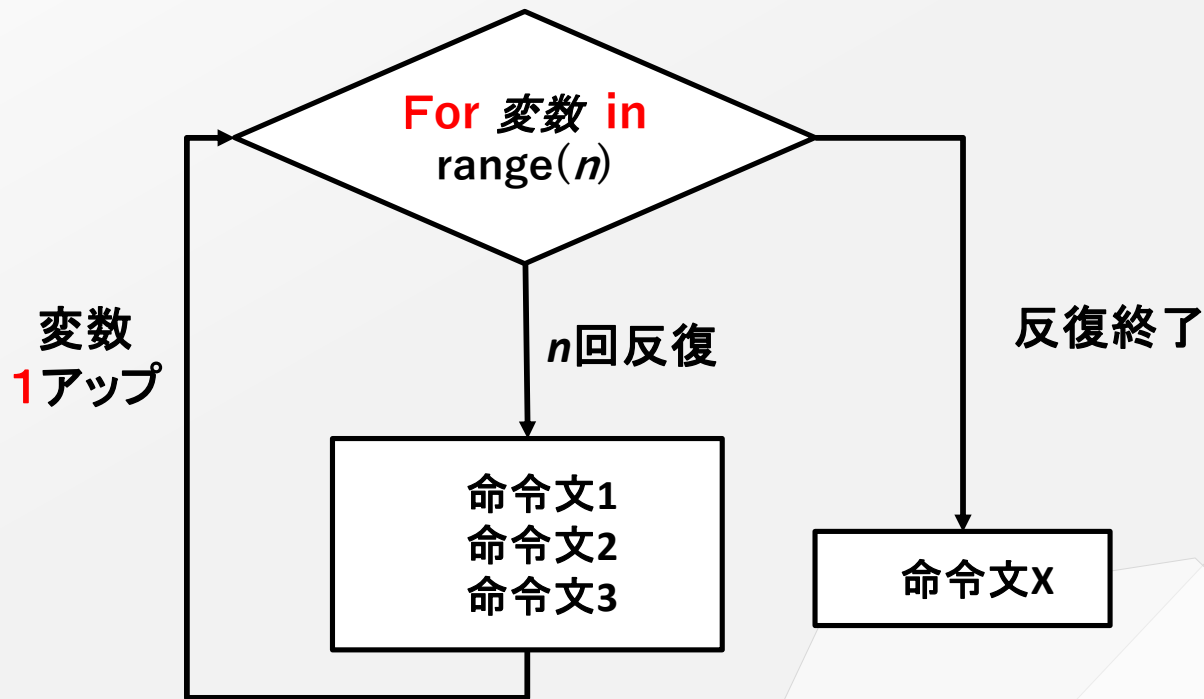
命令文X # インデントのない命令文Xはfor文の後実行



While・for文はC言語やJavaScriptでも使う反復文

Pythonの反復： for in

- forの反復処理をフローチャートで書くと



Pythonの反復： for in

➤ For文で、10までの奇数を出力

```
for count in range(10):    # 0から9まで。合計10回
    if count % 2 == 0:     # 偶数ならスキップ
        continue
    print(count)
print('for 終了')
```

```
1
3
5
7
9
for 終了
```

- ✓ countは最初『0』
- ✓ 次のループでcountは1アップ
- ✓ countが『10-1』（10回）でループ終了

（whileの奇数出力の例と比べよう）

range関数

■ range (開始値, 終了値, ステップ)

list(range(5))

list(range(0))

list(range(2, 5))

list(range(-4, 0))

list(range(2, 9, 2))

list(range(9, 2, -3))

✓ **開始値**の省略値は『0』

✓ **ステップ**の省略値は『1』

Pythonの反復： for in

➤ range関数の引数を工夫して、10までの奇数を出力

```
▶ for count in range(1,10,2):  
    print(count)  
print('for 終了')
```

```
1  
3  
5  
7  
9  
for 終了
```

✓ 開始値を1、ステップを2

✓ 1行で書ける！

Pythonの反復： for in

- **for 変数 in list :** list (配列) の要素分繰り返す

forとlist

```
▶ colors = [ '赤', '緑', '青' ]  
for color in colors:  
    print(color)
```

赤
緑
青

数値listの平均値を計算する

```
▶ scores = [ 68, 22, 90, 77, 81 ]  
sum = 0  
for e in scores:  
    sum += e  
print( sum / len(scores))
```

67.6

- ✓ 最初のループは『赤』
- ✓ 次のループで『緑』
- ✓ 最後の『青』でforループ終了

Pythonの反復： for in

- **for 変数 in string** : 文字列の分繰り返す

```
▶ words = 'python'
for letter in words : # wordsを1文字ずつ反復
    print(letter)
```

p
y
t
h
o
n