

# 9. 遷移関数、 知的なゲーム世界の実現

(人工知能)

URL: <https://www.kkaneko.jp/ai/mi/index.html>

金子邦彦





① 遷移関数の有用性

② プログラムでできること

# アウトライン

1. 人工知能の種類
2. 遷移関数
3. コンピュータとプログラム
4. 状態、行動、遷移をコンピュータで実現
5. コンピュータプレイヤーがゲームに参加

## 9-1 人工知能の種類

## 人工知能

### 機械学習

**データを用いて自ら学習できる  
人工知能**

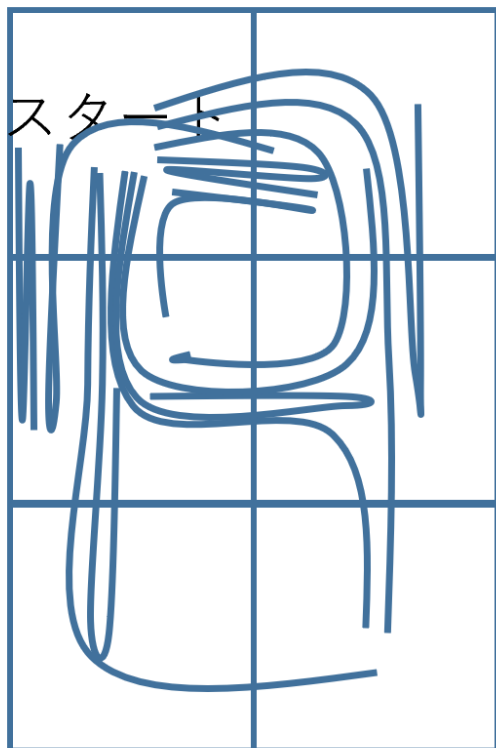
データを与えて学習させることで、  
より正確に予測や分析や合成ができるように

### 知的な IT システム

**人間が書いた行動や知識を用  
いて、意思決定や問題解決を行  
う人工知能**

人間が直接プログラムするため、判別性は  
高いが、柔軟性や汎用性は低いと考える人も

6の小部屋. 3回動くとどこに  
たどり着くか



```
import itertools
import sys

def move(x, y, r):
    success = False
    if (r == 1) and (x < 1):
        x = x + 1
        success = True
    if (r == 2) and (x > 0):
        x = x - 1
        success = True
    if (r == 3) and (y < 2):
        y = y + 1
        success = True
    if (r == 4) and (y > 0):
        y = y - 1
        success = True

    return(x, y, success)

nsteps = 3
seq = [1, 2, 3, 4]
success = False
for j in list(itertools.product(seq, repeat=nsteps)):
    x, y = 0, 0
    for i in j:
        x, y, success = move(x, y, i)
        if(not(success)):
            break

    if(success):
        print("%s %d %d" % (str(j), x, y))

nsteps = 3
seq = [1, 2, 3, 4]
success = False
for j in list(itertools.product(seq, repeat=nsteps)):
    x, y = 0, 0
    for i in j:
        x, y, success = move(x, y, i)
        if(not(success)):
            break

    if(success):
        print("%s %d %d" % (str(j), x, y))
```

上下左右に  
動くことができる  
という行動を  
プログラム化

## 得られた結果（抜粋）

```
(1, 2, 1) 1 0
(1, 2, 3) 0 1
(1, 3, 2) 0 1
(1, 3, 3) 1 2
(1, 3, 4) 1 0
(3, 1, 2) 0 1
```

## 9-2 遷移関数

# 遷移関数



- **遷移関数は、特定の行動を取ったときに現在の状態がどのように変化するかを定める規則。**
- AIが「**ある行動をとると結果としてどのような状態になるか**」を理解するために使用



# 遷移関数を学ぶ意義



- **AIの理解が進む**

AIは、**遷移関数**を用いて、**可能な全ての変化後の状態**を検討し、その結果を評価し、**最善の行動**を選択する。

- **問題解決スキルが向上する**

遷移関数は、AIだけでなく、論理的な問題解決全般に役立つ。  
「**現在の状態がどのように変化するか**」を理解することで、  
解決策の立案、最善の結果の達成につながる。

**遷移関数**を学ぶことは、AIの理解を深めるだけでなく、問題解決スキルの向上にも有用

# 遷移関数を理解するために



## ① 遷移関数

**遷移関数は、特定の行動を取ったときに現在の状態がどのように変化するかを定める規則である。**

## ② 遷移関数を学ぶ意義

遷移関数の理解はAIの働きや、問題解決スキルの向上に**役立つ**。

## ③ 今から、遷移関数を具体的に理解していく

今から、「**エレベーターの遷移関数**」や「**21ゲームの遷移関数**」を学び、それらがどのように動作するかを理解することで、**遷移関数の働きを具体的に理解**する。

# エレベーターの遷移関数



- **状態**

エレベーターの現在の階数

- **行動**

エレベーターを上に移動させる（行動 1）か、  
下に移動させる（行動 2）か、  
その場に留まる（行動 3）

- **遷移関数**

行動 1 :    もし、現在が最上階でなければ  
                 階数 = 階数 + 1    . . . 階数が 1 増える

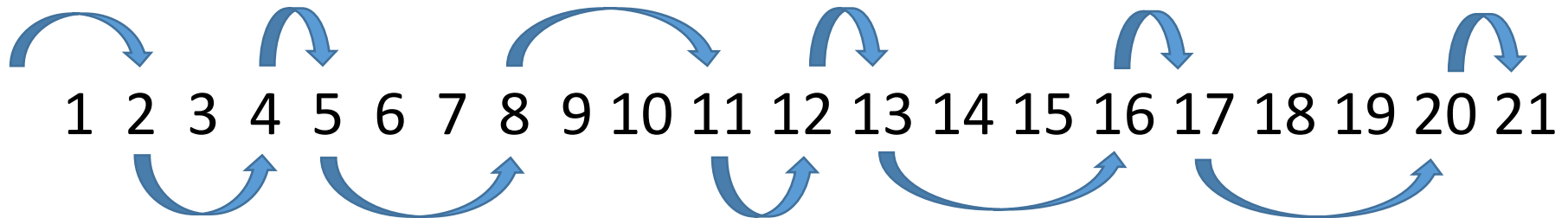
行動 2 :    もし、現在が最下階でなければ  
                 階数 = 階数 - 1    . . . 階数が 1 減る

行動 3 :    階数 = 階数    . . . 階数は変わらない

## 21 ゲームのルール



- 2人のゲーム
- ゲームは「0」から開始
- 2人で交互に、現在の数に、**1または2または3**を足すことができる。これにより、数が増える。
- **21**を言った人が負けとする



## 21 ゲームの対戦イメージ



最初は0

先攻

後攻

1, 2

3, 4

5

6, 7, 8

9, 10, 11

12

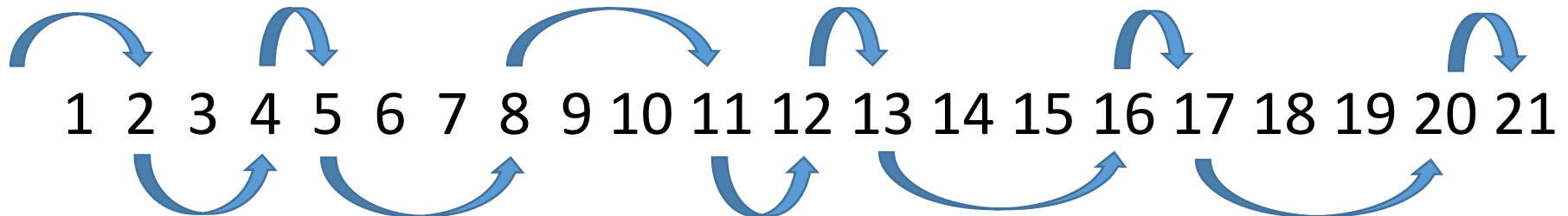
13

14, 15, 16

17

18, 19, 20

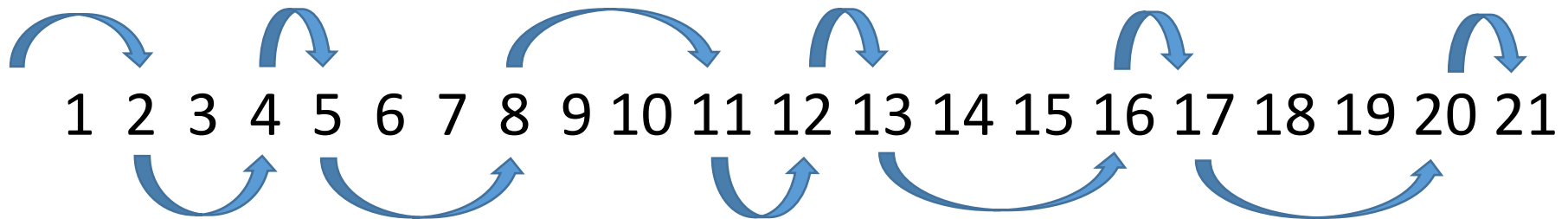
21 (負け)



## 21 ゲームの魅力



- 21ゲームの面白さは、**戦略的な思考**が必要となるところにある。
- プレイヤーの目的は、**自分の番で21を作り出さないようにしつつ、相手を21を作り出させる**ことである。
- そのため、プレイヤーは戦略的に数字 1, 2, 3 を選ばなければならない。



# 21 ゲームの遷移関数



- 状態

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,  
19, 20, 21      ※ 最初は 0 である

- 行動

1 を足す (行動 1) 、 2 を足す (行動 2) 、 3 を足す  
(行動 3)

- 遷移関数

行動 1 :      (数 < 21 のときのみ可能)

数 = 数 + 1      . . . 数が 1 増える

行動 2 :      (数 < 21 のときのみ可能)

数 = 数 + 2      . . . 数が 2 増える

行動 3 :      (数 < 21 のときのみ可能)

数 = 数 + 3      . . . 数が 3 増える

- **遷移関数**は、**特定の行動**を取ったときに**現在の状態**が**どのように変化するか**を定める**規則**である。
- **遷移関数**は、AIだけでなく、論理的な問題解決全般に適用可能。AIの動作原理の理解や、問題解決能力の上達に重要。
- **状態**は、**具体的な問題や状況**を表す。
- エレベーターでは、**現在の階数**が**状態**で、**エレベーターの上昇、下降、停止**が**行動**となる。
- 21ゲームでは、**現在の数**が**状態**で、**1、2、3の追加**が**行動**となる。
- AIは、**遷移関数**を用いて、**可能な全ての変化後の状態**を検討し、その結果を評価し、**最善の行動**を選択する。



## 9-3 コンピュータとプログラム

# コンピュータとプログラム



- **コンピュータ**は, **プログラム**に従って動作
- **プログラム**は, **コンピュータ**に指示を出し, 所定の作業を遂行させる

```
x1="100%" y1="0%" x2="0%" y2="100%"
color="#06101F" offset="0%">
color="#1D304B" offset="100%">
stroke="black" stroke-width="450" rx="8" fill="url(#g1)"/>
</stroke>
<rect height="96" viewBox="0 0 96 96" width="96"
  <gradient x1="87.565%" y1="15.875%"
  <stop stop-color="#FFF" stop-opacity="1" offset="0%">
  <stop stop-color="#FFF" offset="100%">
  </gradient>
  x="-500%" y="-500%" width="1000%" height="1000%"
  offset dy="16" in="SourceAlpha" result="alpha"/>
  GaussianBlur stdDeviation="24" result="blur"/>
  colorMatrix values="0 0 0 0 0.2"/>
  rule="evenodd">
```

# プログラミング



- **プログラム**を設計し作成する  
プロセス（プログラミング）  
は、創造的な活動
- アイデアを形にできることが、  
**プログラミング**の魅力



# ソースコード

- ソースコードは、プログラミング言語で書かれたプログラムのもの
- 人間も読み書き，編集できる
- ソースコードにより，プログラムの動作を理解し，必要に応じて改変できる



```
function() { //is the element hidden?
  if (!t.is(':visible')) {
    //it became hidden
    t.appeared = false;
    return;
  }

  //is the element inside the visible window?
  var a = w.scrollLeft();
  var b = w.scrollTop();
  var o = t.offset();
  var x = o.left;
  var y = o.top;

  var ax = settings.accX;
  var ay = settings.accY;
  var th = t.height();
  var wh = w.height();
  var tw = t.width();
  var ww = w.width();

  if (y + th + ay >= b &&
      y <= b + wh + ay &&
      x + tw + ax >= a &&
      x <= a + ww + ax) {

    //trigger the custom event
    if (!t.appeared) t.trigger('appear', settings.data);

  } else {

    //it scrolled out of view
    t.appeared = false;
  }
};

//create a modified fn with some additional logic
var modifiedFn = function() {

  //mark the element as visible
  t.appeared = true;

  //is this supposed to happen only once?
  if (settings.one) {

    //remove the check
    w.unbind('scroll', check);
    var i = $.inArray(check, $.fn.appear.checks);
    if (i >= 0) $.fn.appear.checks.splice(i, 1);

  }

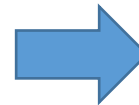
  //trigger the original fn
  fn.apply(this, arguments);
}
```

# プログラミングの目的



- **プログラム**は、**コンピュータ**に指示を出し、所定の作業を遂行させる
- 複雑な作業も**自動化**し、効率化することが可能

```
a = [200, 400, 300]
for i in a:
    print (i * 1.08)
```



```
216.0
432.0
324.0
```

**Python プログラム**の  
ソースコード

**プログラム**の  
実行結果

## 9-4 状態、行動、遷移関数をコンピュータで実現

# いまから行うこと



- 21 ゲームの**状態、行動、遷移関数**を,  
コンピュータの**プログラム**として書き,  
コンピュータのゲームとして動くようにする
- プログラミング言語として Python を使用する

# 21 ゲームの遷移関数



- 状態

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,  
19, 20, 21      ※ 最初は 0 である

- 行動

1 を足す (行動 1) 、 2 を足す (行動 2) 、 3 を足す  
(行動 3)

- 遷移関数

行動 1 :      (数 < 21 のときのみ可能)

数 = 数 + 1      . . . 数が 1 増える

行動 2 :      (数 < 21 のときのみ可能)

数 = 数 + 2      . . . 数が 2 増える

行動 3 :      (数 < 21 のときのみ可能)

数 = 数 + 3      . . . 数が 3 増える



a は数  
r は行動番号

(数 < 21 のときのみ可能)

数 = 数 + 1

・・・数が 1 増える

行動 1



if (r == 1) and (a < 21):  
a = a + 1

選んだ行動が 1 のときは、  
a < 21 を調べ、  
a を a + 1 に変化

プログラムの  
ソースコード

## 演習

**21 ゲームの状態、行動、遷移関数**  
を実現する Python 言語プログラム  
を動作させ、理解を深める

### 【トピックス】

- trinketでのプログラム実行
- 状態、行動、遷移関数

# trinket でのプログラム実行



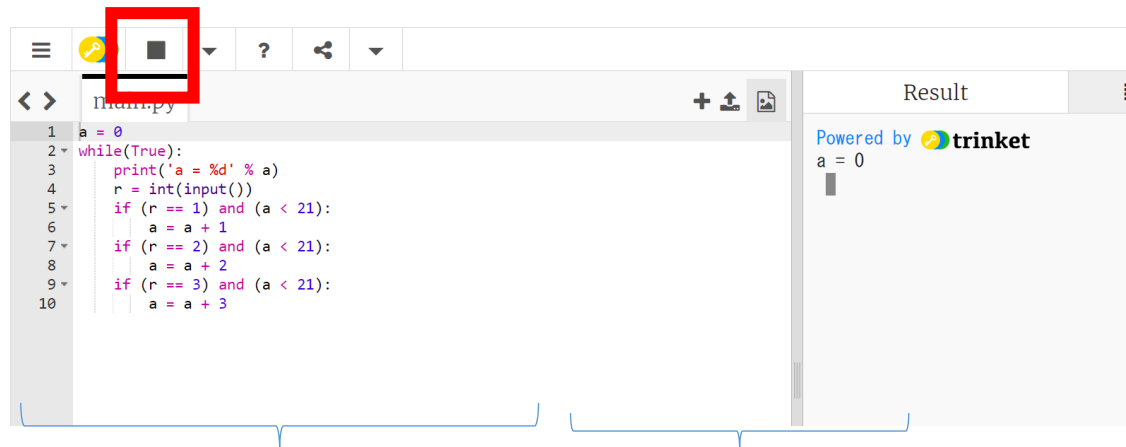
- URL: <https://trinket.io/python/58bb317816>

違うプログラムには違う URL が割り当てられる

実行、STOP ボタン

## Put Interactive Python Anywhere on the Web

Customize the code below and Share!



ソースコードの  
編集画面

実行結果

- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを書き替えて再度実行することも可能

# ソースコード



```
a = 0
```

} 最初 a の値は 0

```
while(True):
```

```
    print('a = %d' % a)
```

```
    r = int(input())
```

```
    if (r == 1) and (a < 21):
```

```
        a = a + 1
```

```
    if (r == 2) and (a < 21):
```

```
        a = a + 2
```

```
    if (r == 3) and (a < 21):
```

```
        a = a + 3
```

行動 1

行動 2

行動 3

ユーザーから入力を受け取り（1、2、または3を足す）、それが21以下であればその数を足し、新しい値を出力する。

# 実行開始後の手順



① キーボードで、1 + Enter、2 + Enter、3 + Enter の操作を続ける

1 : 行動 1    2 : 行動 2    3 : 行動 3

② 画面を確認する。2 1 になったらゲームは終わり。「**Stop**」をクリックして終了。

The screenshot shows the Trinket IDE interface. On the left, the code editor displays a Python script named `main.py`. The code is as follows:

```
1 a = 0
2 while(True):
3     print('a = %d' % a)
4     r = int(input())
5     if (r == 1) and (a < 21):
6         a = a + 1
7     if (r == 2) and (a < 21):
8         a = a + 2
9     if (r == 3) and (a < 21):
10        a = a + 3
```

On the right, the 'Result' tab is active, showing the output of the program. It starts with 'Powered by trinket' and then displays a sequence of values for 'a' and the input 'r'.

a	r
0	
2	1
3	
6	3
9	
11	1
12	
13	3
16	
19	2
21	
21	2
21	
21	2
21	

# まとめ



21 ゲームの**状態、行動、遷移関数**を、  
コンピュータの**プログラム**として書き、  
コンピュータのゲームとして動くようにする

## 遷移関数

- 行動 1 :** (数 < 2 1 のときのみ可能)  
数 = 数 + 1 . . . 数が 1 増える
- 行動 2 :** (数 < 2 1 のときのみ可能)  
数 = 数 + 2 . . . 数が 2 増える
- 行動 3 :** (数 < 2 1 のときのみ可能)  
数 = 数 + 3 . . . 数が 3 増える



Result	Instructions
Powered by  trinket	
a = 0	
2	
a = 2	
1	
a = 3	
3	
a = 6	
3	
a = 9	
2	
a = 11	
1	
a = 12	
1	
a = 13	
3	
a = 16	
3	
a = 19	
2	
a = 21	
2	
a = 21	
2	
a = 21	
2	
a = 21	

## 9-5 コンピュータ・プレイヤー がゲームに参加

## はじめに

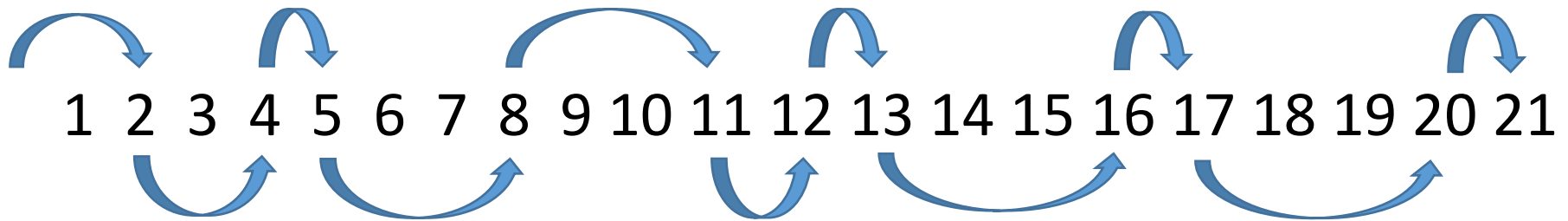
- コンピュータを, ゲームのプレイヤーとして振舞わせてみる
- コンピュータは知的に振舞う



# 後攻は必勝

- 後攻での必勝のやり方

**後攻**は4, 8, 12, 16, 20になるように数を増やす



# 後攻での必勝のやり方



今の数を  $a$  とする

- $a = 1$  のとき, 行動 **3** ( $a$  を 3 増やす)
- $a = 2$  のとき, 行動 **2** ( $a$  を 2 増やす)
- $a = 3$  のとき, 行動 **1** ( $a$  を 1 増やす)
- $a = 5$  のとき, 行動 **3** ( $a$  を 3 増やす)
- 
- 
- 

(続く)

# 後攻での必勝のやり方



今の数を **a** とする

- **a = 1** のとき, 行動 **3**
- **a = 2** のとき, 行動 **2**
- **a = 3** のとき, 行動 **1**
- **a = 5** のとき, 行動 **3**
- **a = 6** のとき, 行動 **2**
- **a = 7** のとき, 行動 **1**
- **a = 9** のとき, 行動 **3**
- **a = 10** のとき, 行動 **2**
- **a = 11** のとき, 行動 **1**

- **a = 13** のとき, 行動 **3**
- **a = 14** のとき, 行動 **2**
- **a = 15** のとき, 行動 **1**
- **a = 17** のとき, 行動 **3**
- **a = 18** のとき, 行動 **2**
- **a = 19** のとき, 行動 **1**

※ このやり方のとき,

後攻は,  $a = 4, 8, 12, 16,$   
20 で自分の手番がくるこ  
とはない

今の数を **a** とする

**a = 1** のとき, 行動 **3**



行動番号を **r** とする

```
if (a == 1):  
    r = 3
```

プログラムの  
ソースコード

## 演習

**21 ゲームでコンピュータープレイヤーを実現する Python 言語プログラムを動作させ、理解を深める**

### 【トピックス】

- **trinketでのプログラム実行**
- **状態、行動、遷移関数**

# ソースコード



```
def computer(a):  
    if (a == 1):  
        return 3  
    elif (a == 2):  
        return 2  
    elif (a == 3):  
        return 1  
    elif (a == 5):  
        return 3  
    elif (a == 6):  
        return 2  
    elif (a == 7):  
        return 1  
    elif (a == 9):  
        return 3  
    elif (a == 10):  
        return 2  
    elif (a == 11):  
        return 1  
    elif (a == 13):  
        return 3  
    elif (a == 14):  
        return 2  
    elif (a == 15):  
        return 1  
    elif (a == 17):  
        return 3  
    elif (a == 18):  
        return 2  
    elif (a == 19):  
        return 1  
    else:  
        return 0
```

コンピュータ  
が自分の使う  
行動を  
決める

```
def move(a, r):  
    if (r == 1) and (a < 21):  
        a = a + 1  
    if (r == 2) and (a < 21):  
        a = a + 2  
    if (r == 3) and (a < 21):  
        a = a + 3  
  
    return a
```

```
a = 0  
while(True):  
    print('a = %d' % a)  
    r = int(input())  
    a = move(a, r)  
    print('a = %d' % a)  
    r = computer(a)  
    print('computer: %d' % r)  
    a = move(a, r)
```

行動

最初 a の  
値は 0

表示  
など

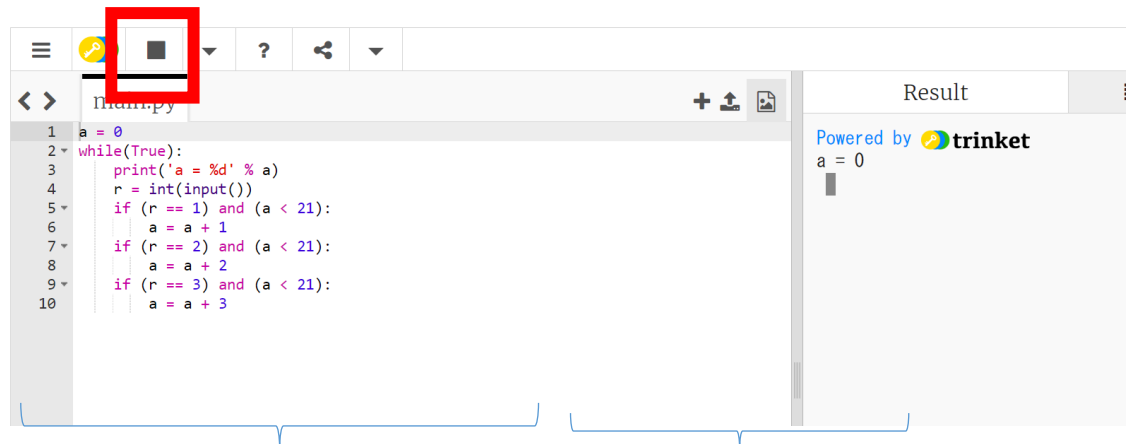
# trinket でのプログラム実行



- URL: <https://trinket.io/python/5468e2724a>
  - 違うプログラムには違う URL が割り当てられる
- 実行、STOP ボタン

## Put Interactive Python Anywhere on the Web

Customize the code below and Share!



ソースコードの  
編集画面

実行結果

- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを書き替えて再度実行することも可能

# 実行開始後の手順



① キーボードで、1 + Enter、2 + Enter、3 + Enter の操作を続ける

1 : 行動 1    2 : 行動 2    3 : 行動 3

② 今度はコンピュータが対戦相手である。画面を確認する。2 1 になったらゲームは終わり。「Stop」をクリックして終了。

```
main.py
21     return 3
22 elif( a == 14 ):
23     return 2
24 elif( a == 15 ):
25     return 1
26 elif( a == 17 ):
27     return 3
28 elif( a == 18 ):
29     return 2
30 elif( a == 19 ):
31     return 1
32 else:
33     return 0
34
35 def move(a, r):
36     if (r == 1) and (a < 21):
37         a = a + 1
38     if (r == 2) and (a < 21):
39         a = a + 2
40     if (r == 3) and (a < 21):
41         a = a + 3
42     return a
43
44 a = 0
45 while(True):
46     print('a = %d' % a)
47     r = int(input())
48     a = move(a, r)
49     print('a = %d' % a)
50     r = computer(a)
51     print('computer: %d' % r)
52     a = move(a, r)
53
54
55
```

```
Powered by trinket
a = 0
3
a = 3
computer: 1
a = 4
1
a = 5
computer: 3
a = 8
3
a = 11
computer: 1
a = 12
1
a = 13
computer: 3
a = 16
2
a = 18
computer: 2
a = 20
2
a = 22
computer: 0
a = 22
```



## コンピュータを, ゲームのプレイヤーとして振舞わせてみた

今の数を **a** とする

- **a = 1** のとき, ルール **3**
- **a = 2** のとき, ルール **2**
- **a = 3** のとき, ルール **1**
- **a = 5** のとき, ルール **3**
- **a = 6** のとき, ルール **2**
- **a = 7** のとき, ルール **1**
- **a = 9** のとき, ルール **3**
- **a = 10** のとき, ルール **2**
- **a = 11** のとき, ルール **1**
- **a = 13** のとき, ルール **3**
- **a = 14** のとき, ルール **2**
- **a = 15** のとき, ルール **1**
- **a = 17** のとき, ルール **3**
- **a = 18** のとき, ルール **2**
- **a = 19** のとき, ルール **1**

※ このやり方のとき,  
後攻は, a = 4, 8, 12, 16,  
20 で自分の手番がくるこ  
とはない

後攻での必勝法



```
Powered by trinket
a = 0
3
a = 3
computer: 1
a = 4
1
a = 5
computer: 3
a = 8
3
a = 11
computer: 1
a = 12
1
a = 13
computer: 3
a = 16
2
a = 18
computer: 2
a = 20
2
a = 22
computer: 0
a = 22
```

## ① 遷移関数の有用性

**遷移関数**は、**特定の行動**を取ったときに**現在の状態がどのように変化するか**を定める**規則**である。AIだけでなく、論理的な問題解決全般にも適用可能である。

## ② プログラムでできること

**プログラム**を用いて、**ゲームの世界**（**状態**、**行動**、**遷移関数**）や**コンピュータープレイヤー**をコンピュータ内に実装することができる。

- **遷移関数**は、**特定の行動**を取ったときに**現在の状態**が**どのように変化するか**を定める**規則**である。
- **遷移関数**は、AIの動作原理の理解や、問題解決能力の上達に重要
- 「21ゲーム」では、**現在の数**が**状態**、1の追加と2の追加と3の追加が**行動**となる。**遷移関数**は**行動**により**状態がどう変化するか**を定める。