11. プログラミングによる人工知能の実現

（人工知能）

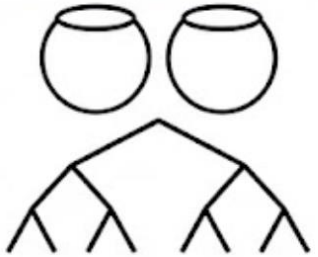
金子邦彦

「人工知能」第１１回の内容

プログラミングの基本機能で、AIの状態・行動・遷移を表し、組み合わせて探索を実現する

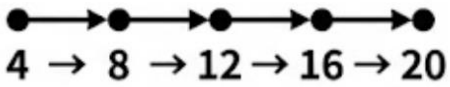
2つの水差し問題（総当たり型）

8通りの行動を
すべて試す
→目標の水量に
達する手順を発見



21ゲーム（必勝法型）

後手は自分の手番で合計を
常に**4の倍数**に保てば
必ず勝てる



組み合わせると → 探索

状態
＝変数

遷移関数
＝関数

プログラミングの5つの基本機能：
入力/出力/順次実行/条件分岐/繰り返し

11-1. コンピュータとプログラム

コンピュータとプログラム

プログラミングは、自分のアイデアをコンピュータ上で形にする創造的な手段

コンピュータの5つの基本動作

入力
（情報をもらう）

出力
（結果を見せる）

順次実行
（書いた順に実行）

条件分岐
（条件で処理を変える）

繰り返し
（決めた回数くり返す）

プログラムの動き

ソースコード → コンピュータが処理 → 結果

人間が読める言葉で書いた命令の集まり
＝ 読める・書ける・直せる

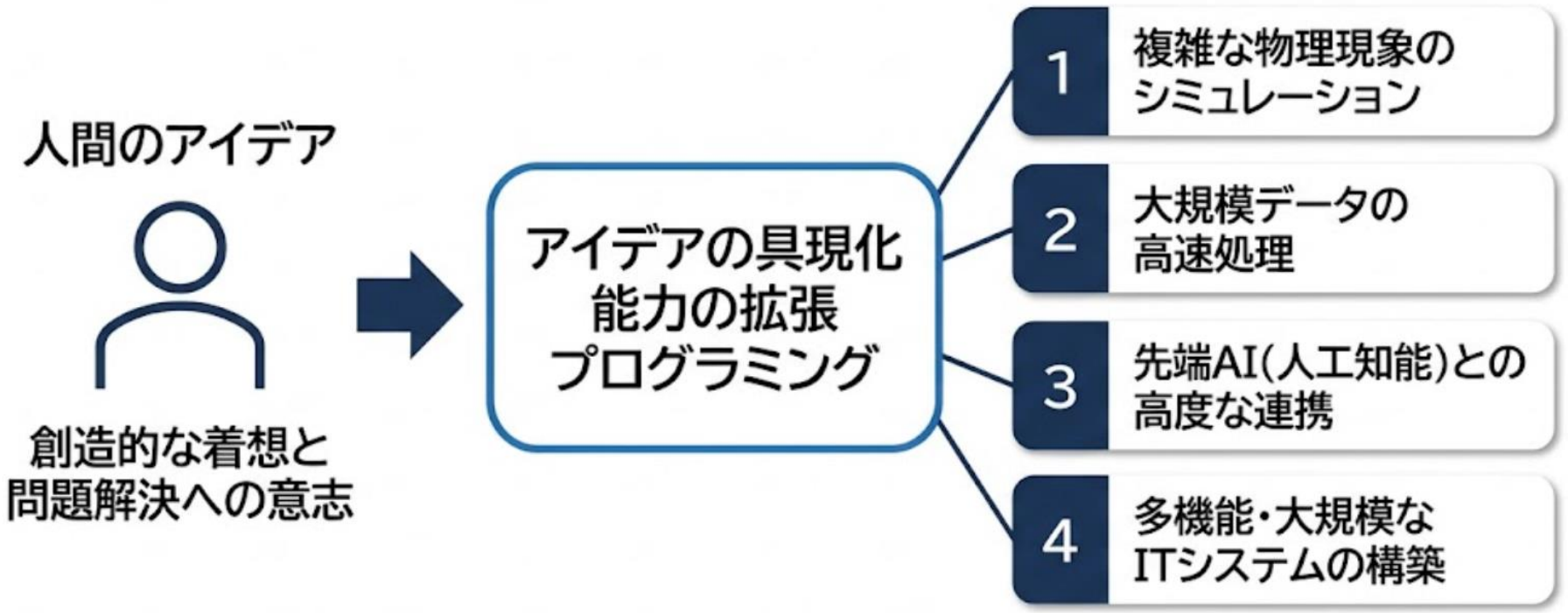
Python の3つの利点

①文法がシンプル
＝ print / if・else / for・while
字下げ（インデント）でブロックを示す

②豊富なライブラリ
＝ データ分析・AI・Web・自動化

③柔軟性 ＝ 小規模～大規模まで同じ言語で

プログラミングで何ができるのか

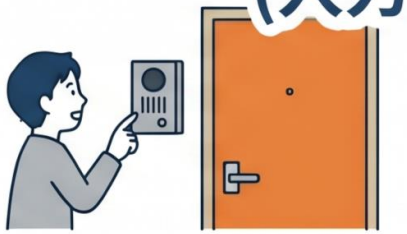


誤解:単なる『覚えて、問題を解くだけの勉強』

正解:『自分のアイデアを形にする』創造的な作業

コンピュータはプログラムの指示通りに動く

キーボードなどから
情報をもらう (入力)



順次実行

結果を相手に見せる
(出力)

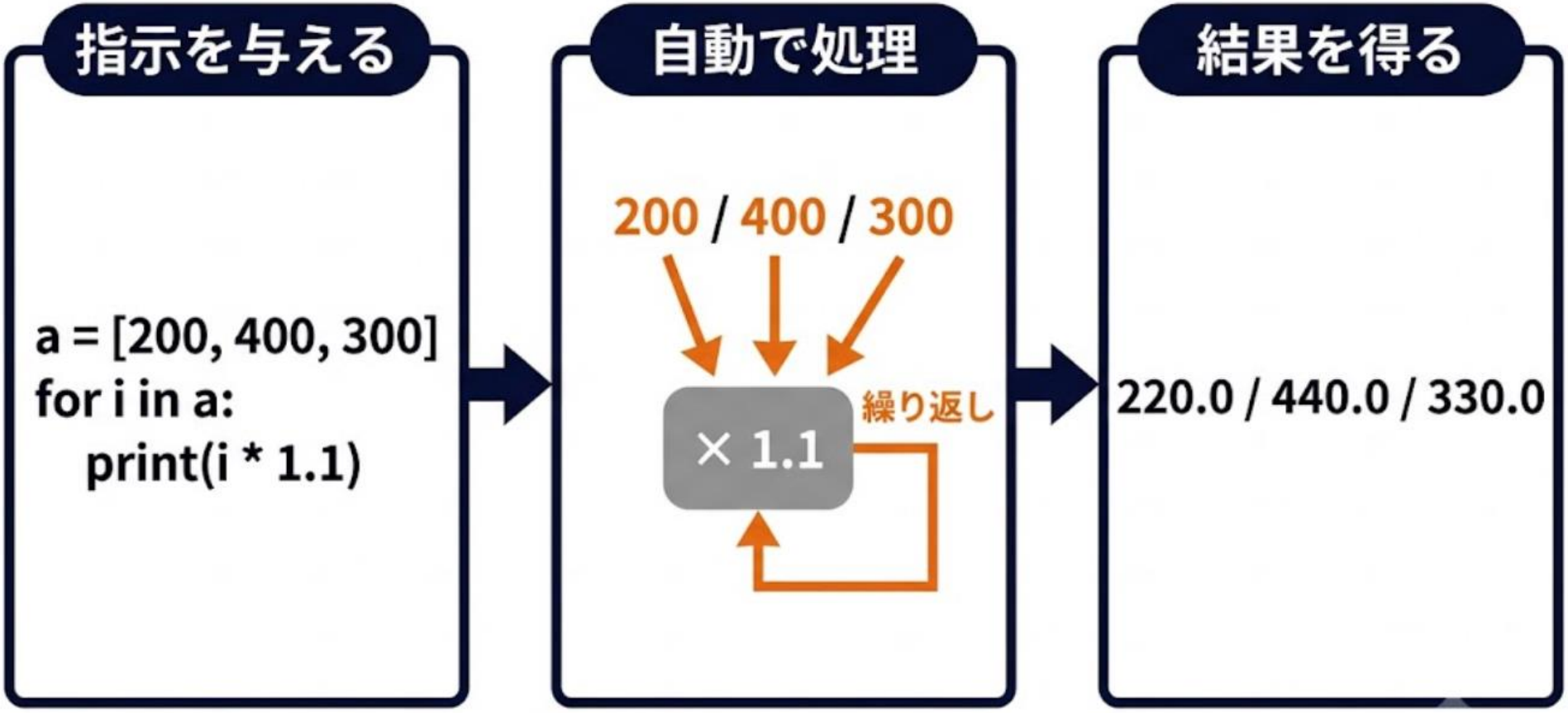
もし雨なら傘を持つ、
晴れなら持たない (条件分岐)

決めた回数だけ
同じ動作をくり返す
(繰り返し)

どの言語も
働きは同じ、
書き方が違う

Pythonは
書き方の見た目
整理されていて
読みやすい

プログラムとは ― 命令を書いた手順の集まり



だから複雑な作業も【自動化・効率化】できる

ソースコード

ソースコードは、人間が読み書きできる言葉で書かれたプログラムである

Python 例

```
price = 100
tax = price * 0.1
print(price + tax)
```

人間が読める言葉
（プログラミング言語）で
書かれている

編集・修正

3つの特徴

1. 読める … 何をする処理か理解できる
2. 書ける … 人が新しく作成できる
3. 直せる … 必要に応じて改変できる

改変
変更前：`tax = price * 0.1`
変更後：`tax = price * 0.08`

実行での流れ

Python の特徴

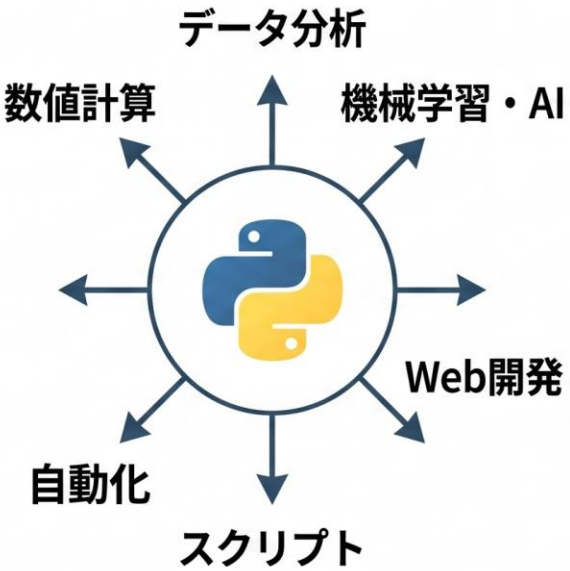
Python には、文法のシンプルさ・ライブラリの豊富さ・柔軟性という利点がある

文法がシンプル・読みやすい

出力	print
条件分岐	if / else
繰り返し (ループ)	for / while

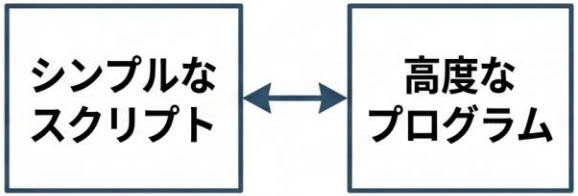
ブロック構造	字下げ（インデント）で示す
	if x > 0:
	  print('正')
	else:
	  print('負以下')

豊富なライブラリ



多岐にわたる分野で利用できる

柔軟性



小規模から大規模まで
同じ言語で書ける

11-2. ２つの水差し

状態・行動・遷移関数

行動を選ぶと、状態が次の状態へ変わる ― これを扱うのが状態・行動・遷移関数の枠組み

２つの水差しの例（今からの説明内容）

２つの水差しで『水の量を２にできるか』をコンピュータに解かせる

問題設定

目標：どちらかの水差しを量2にできるか？

状態の変数化

水差し①の水量
＝変数 x

水差し②の水量
＝変数 y

状態：(x, y) で表す

(0,0) → … →

8つの行動と遷移

水差し①の行動	水差し②の行動
①を満杯	②を満杯
①を空に	②を空に
①で②を満杯に	②で①を満杯に
①を全部②へ	②を全部①へ

例）①で②を満杯に：x → x+y−3, y → 3（適用条件：x+y≧3 かつ y<3）

２つの水差しの例：８つの行動

- 移す側の水が足りなくて、受ける側がまだ満杯にならない場合 → **全部移る**（行動4・行動8）
- 移す側の水が多くて、受ける側が先に満杯になる場合 → **受ける側が満杯になり、残りは元に残る**（行動3・行動7）

２つの水差しの例：状態と行動

大きさの違う2つの水差し

目標：水の量を2にしたい

状態を変数で表す

状態 ＝ (x , y)
- x ＝ 水差し①の水の量
- y ＝ 水差し②の水の量

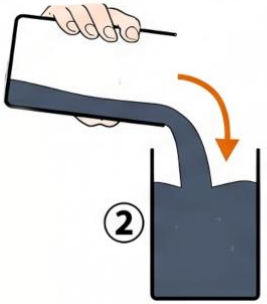
8つの行動

水差し①の行動	水差し②の行動
①を満杯	②を満杯
①を空に	②を空に
①で②を満杯に	②で①を満杯に
①を全部②へ	②を全部①へ

２つの水差しの例：遷移関数と適用条件

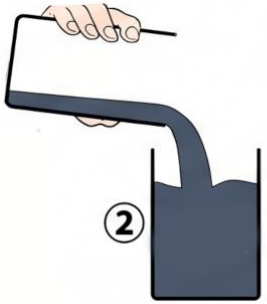
行動の番号	行動の内容	遷移関数	適用条件
行動1	①を満杯	$x \to 4$	$x < 4$
行動2	①を空に	$x \to 0$	$x > 0$
行動3	①で②を満杯	$x \to x + y - 3, y \to 3$	$x + y \geq 3$　$y < 3$
行動4	①を全部②へ	$y \to x + y, x \to 0$	$x + y \leq 3$　$x > 0$
行動5	②を満杯	$y \to 3$	$y < 3$
行動6	②を空	$y \to 0$	$y > 0$
行動7	②で①を満杯に	$x \to 4, y \to x + y - 4$	$x + y \geq 4$　$x < 4$
行動8	②を全部①へ	$x \to x + y, y \to 0$	$x + y \leq 4$　$y > 0$

注ぎ元に水が残るケース



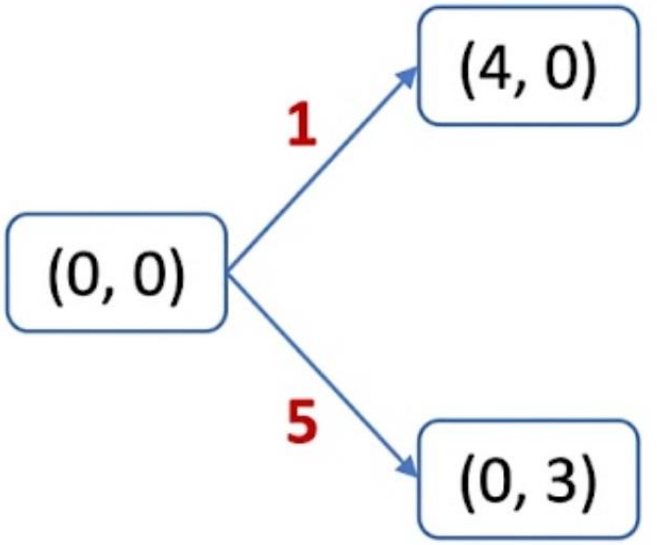
行動3の例
①(4L容器)から②(3L容器)へ注ぐ
②が先に満杯になる
適用条件：x + y ≥ 3 且つ y < 3
結果：y → 3
x → x + y - 3

注ぎ元の水が全部入りきるケース

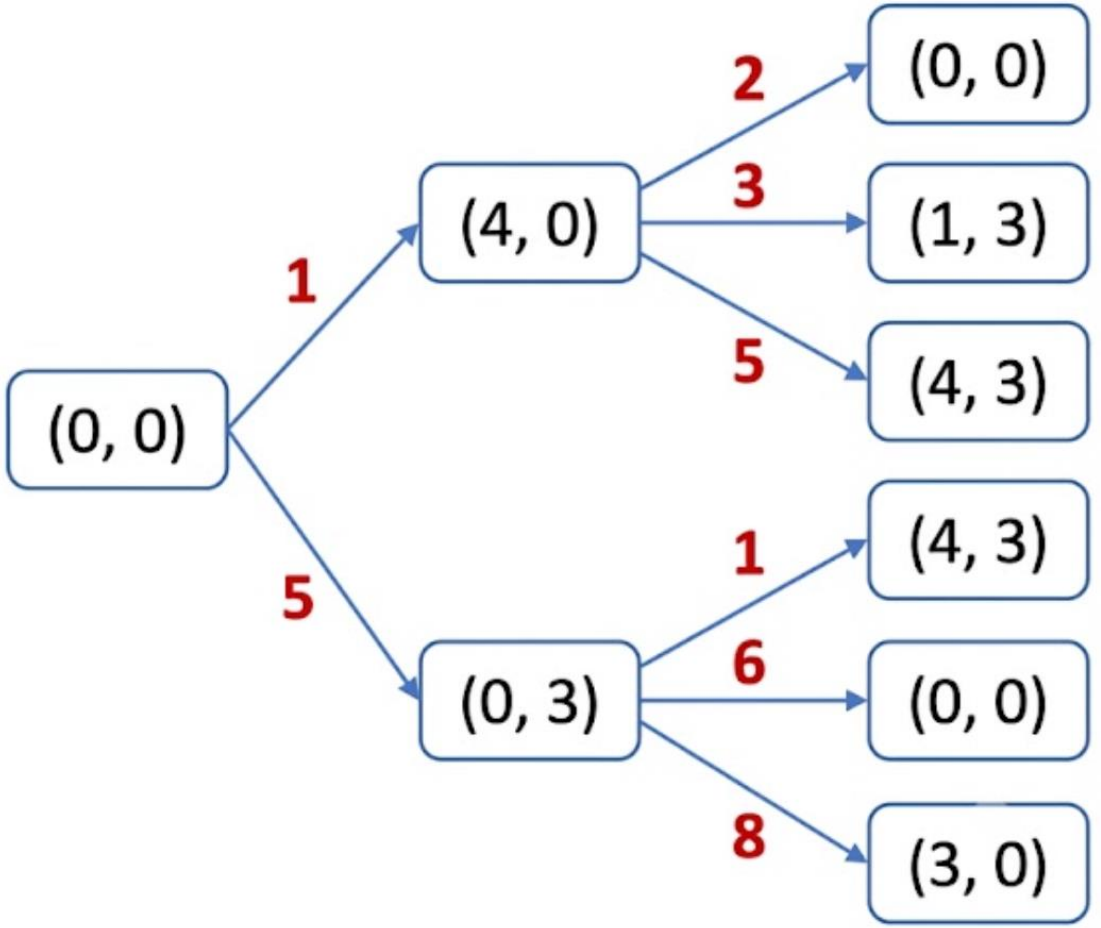


行動4の例
①(4L容器)から②(3L容器)へ注ぐ
①の水が全部入る
適用条件：x + y ≤ 3 且つ x > 0
結果：x → 0
y → x + y

２つの水差しで、行動回数が１回、２回

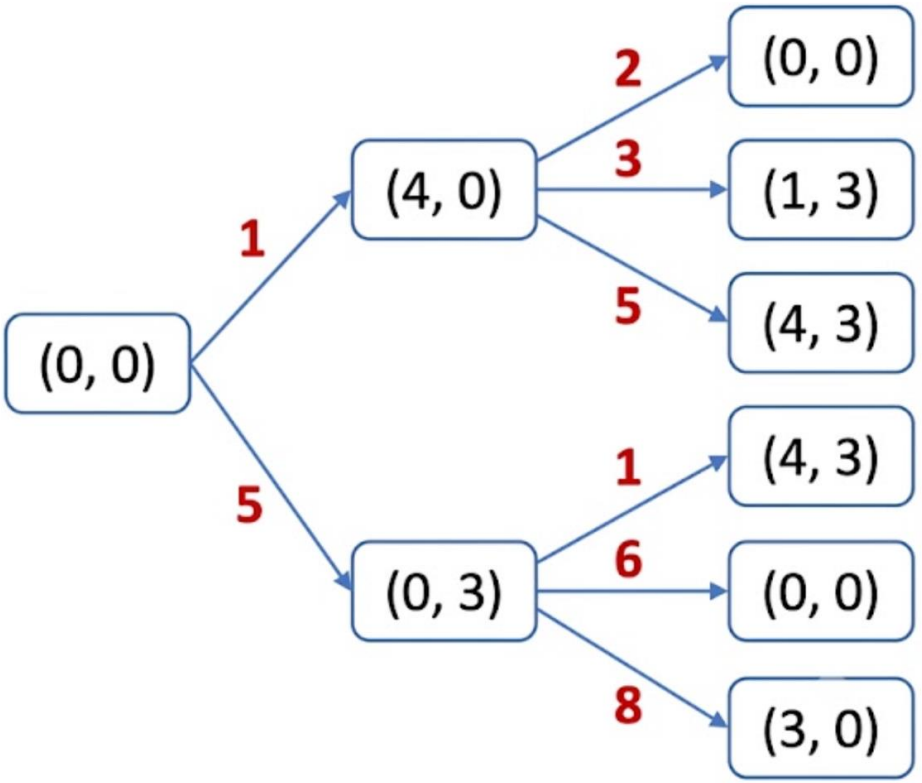


行動１回



行動２回

２階の行動で水の量を「１」にできるかを探索



終了

単純な探索では、

「**水の量１に至る経路（パス）**」を１つでも見つけた時点で探索を終了

11-3. ２１ゲーム

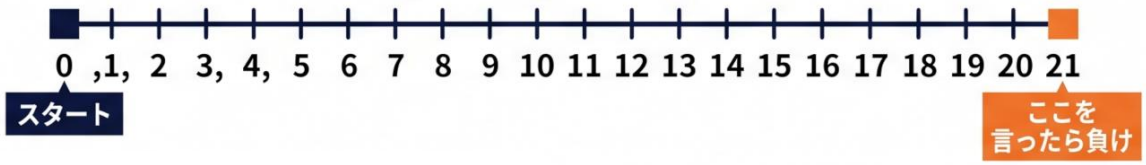
２１ゲーム

0から始めて1～3ずつ足し、21を言った人が負け

ルール

- 0からスタート
- 2人で交互に、自分の番に 1・2・3 のどれかを足す
- 21を言った人が負け

数の流れ



行動

行動1：+1／行動2：+2／行動3：+3

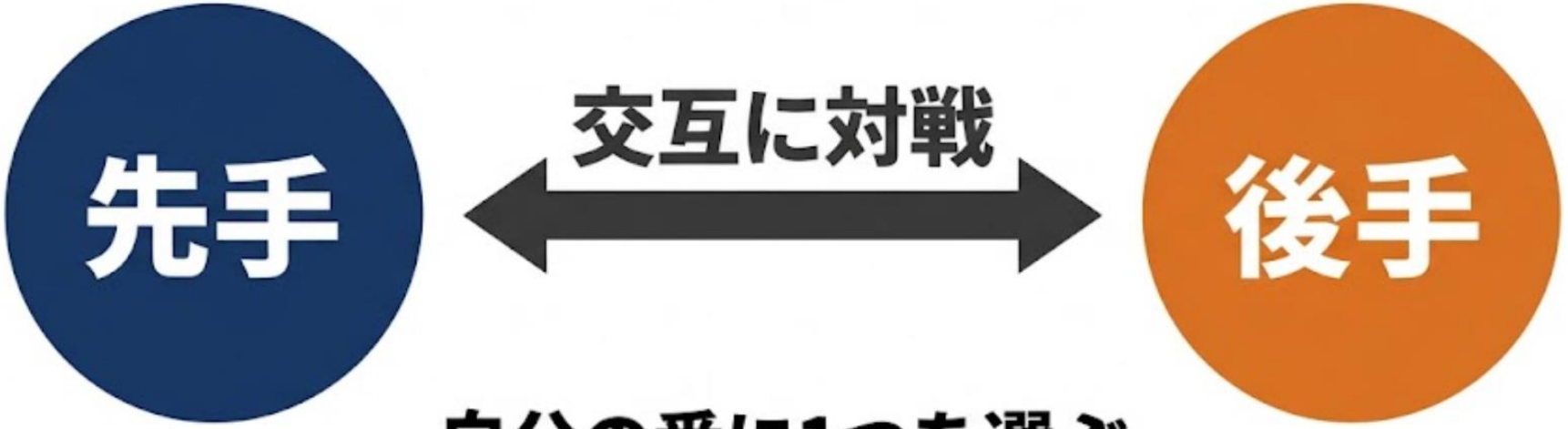
次の数の決まり

次の数 ＝ 今の数 ＋ 1 または 2 または 3

（ただし 今の数 ＜ 21 のときだけ動ける）

２１ゲーム

先手と後手が交互に行動し、毎回
行動1・行動2・行動3 のどれかを選ぶ



自分の番に1つを選ぶ

行動1	行動2	行動3
+1	+2	+3

２１ゲームは後手必勝

勝つコツ：後手は自分のターン終了時の数を『4の倍数』にし、相手を20へ追い込め

先手
自由に１・２・３から選ぶ

相手の手に応じて →

後手
先手の手を見てから選ぶ

後手の応答例

- 先手１→後手３
- 先手２→後手２
- 先手３→後手１

結果：後手は必ず勝てる

※ 具体的な必勝の手順は別の図で説明

２１ゲームは後手必勝

Database Lab.

勝つコツ：自分のターン終了後の数を『4の倍数』にし、相手を20へ追い込め

1ラウンドで「+4」になるように合わせる

相手の手	/	自分の手
+1	→	+3
+2	→	+2
+3	→	+1

合計 +4 を毎ラウンド維持 → 必ず次の『4の倍数』へ到達

論理の核心（後ろから考える）

- 相手に「20」を渡す → 相手は +1 しか選べず「21」を言って負け
- 相手に「16」を渡す → 相手の手に合わせ「20」へ到達可能
- 同様に 12, 8, 4, 0 もすべて敗北位置

開始位置「0」自体が4の倍数 → 完全プレイなら後手必勝

演習

状態と変数

プログラムでは、変化する“状態”を『変数』で表す

2つの水差し

水差し①の水の量 → x

水差し②の水の量 → y

21ゲーム

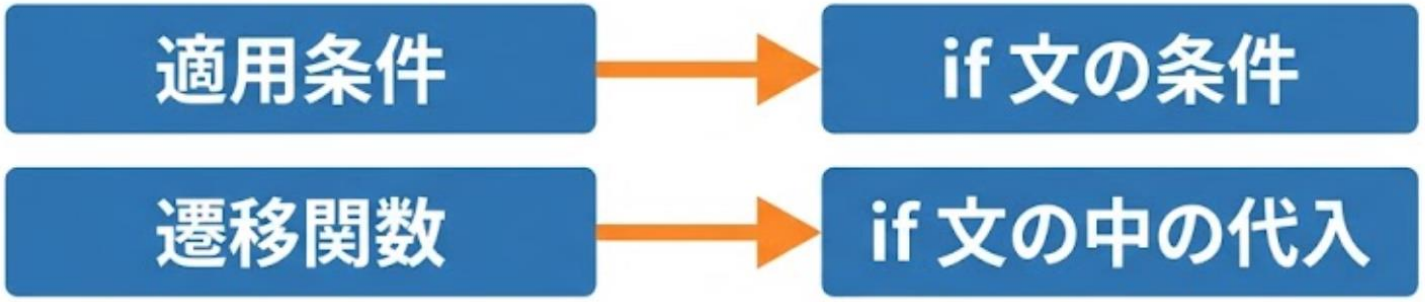
15

今の数 → a

状態（変化する値）＝変数

遷移関数・適用条件と if

遷移関数と適用条件は、そのまま if 文に対応する



２つの水差し / 行動１：①を満杯

適用条件：x < 4 ／ 遷移関数：x → 4

```
if r == 1 and x < CX:
    x = CX
```

CX の値は 4

２１ゲーム / 行動１：数 = 数 + 1

適用条件：数 < 21 ／ 遷移関数：数 = 数 + 1

```
if r == 1 and a < GOAL:
    a = a + 1
```

GOAL の値は 21

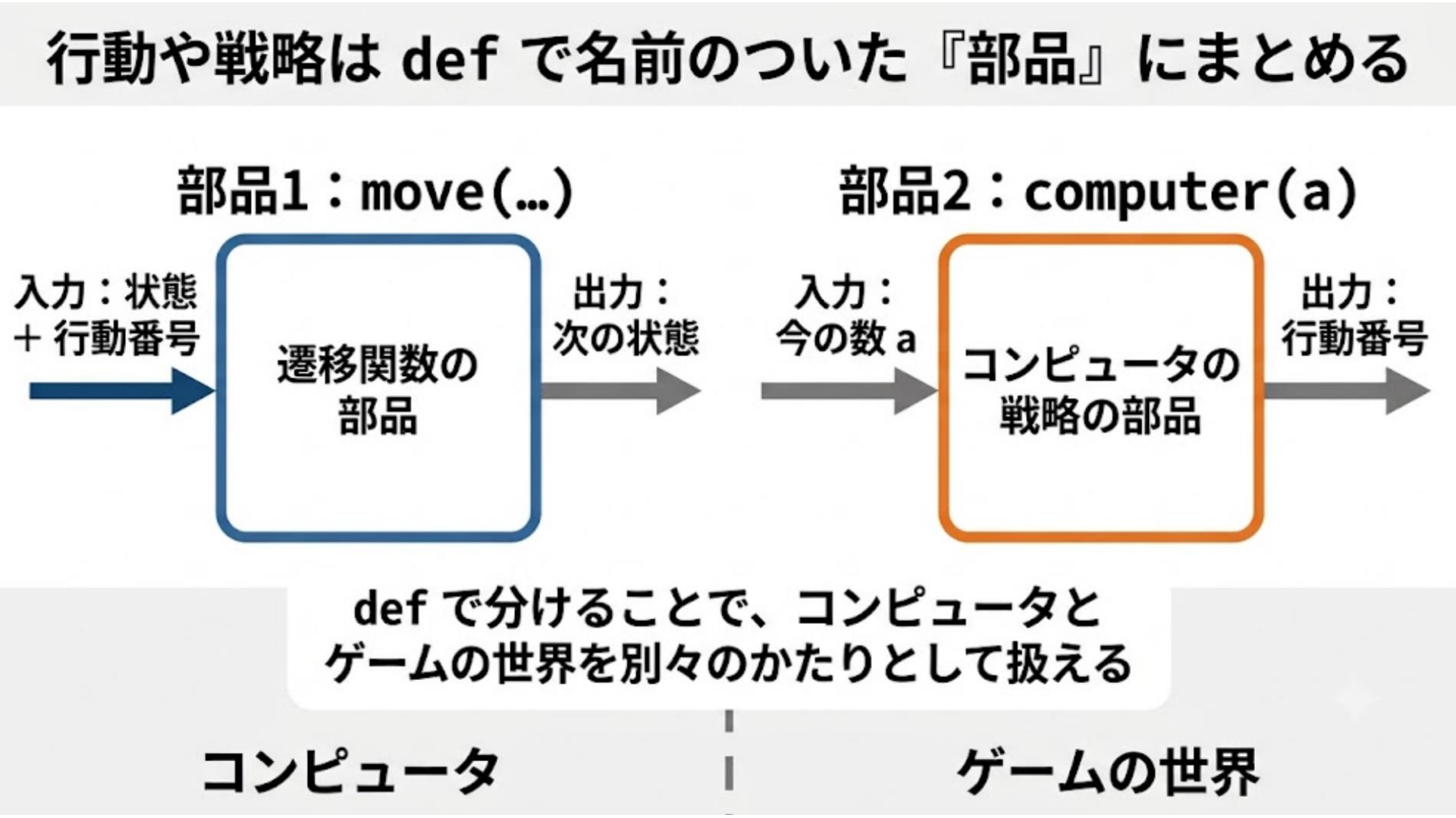
必勝の戦略のプログラム化

日本語で書いた必勝戦略は、そのままプログラムの一行になる

日本語の戦略	そのまま変換	プログラム（一行）
２１ゲームの後手は 自分のターン終了時の数を 『４の倍数』にする	→	r = (4 - a % 4) % 4 ※ r == 0 のときは r = 1 (a は現在の数、r は加える数)

例1）a = 5 → r = 3 → 合計 8（4の倍数）
例2）a = 10 → r = 2 → 合計 12（4の倍数）
例3）a = 8 → r = 0 → 1 → 合計 9（※0なので1にする）

行動・戦略は、プログラム内の「部品」になる



経路（パス）と総当たり（演習１）

2つの水差し（容量4・容量3）で水量2を作れるか——総当たりはすべての経路（パス）を試す

経路（パス）とは何か

行動番号リスト

seq = [1,2,3,4,5,6,7,8]

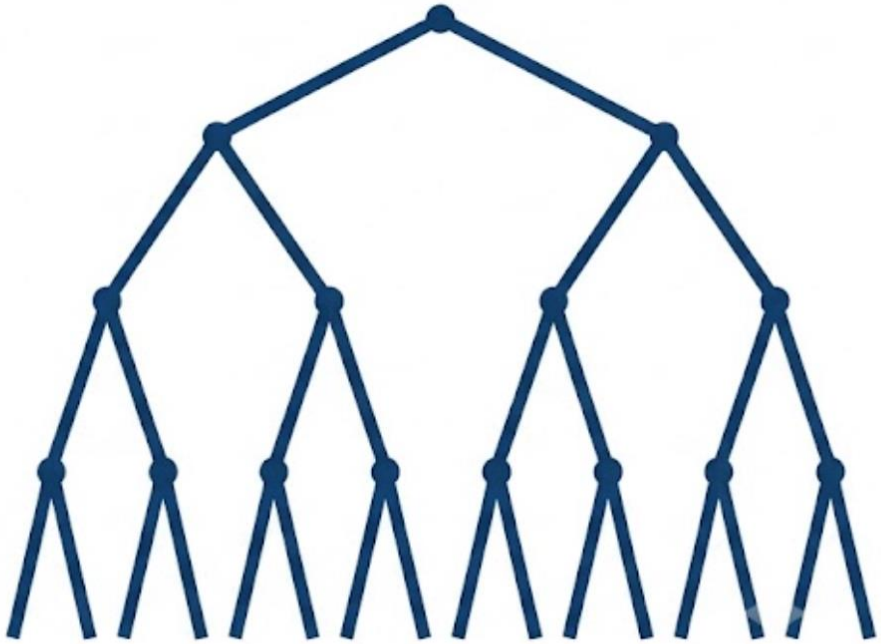
nsteps個の行動を並べたもの
＝ 1つの経路（パス）

[行動5] → [行動8] → [行動5]

これが nsteps=3 の経路の一例

総当たり＝全経路を試す（探索木）

すべての枝（経路）を順に試す＝総当たり



各経路を試して、水量が2になる経路を探す。nsteps を増やすほど経路の総数は増える

実行結果の確認

実行結果を読んで、プログラムが正しく動いているかを確認しよう

演習1：2つの水差し

```
[1, 2] 0 0
```

行動1 → 行動2 の順で実行　x = 0　y = 0

行動を順に行うと
x=0, y=0 になる

演習2：21ゲーム

```
あなたの番(1~3): 3
a = 3
computer: 1, a = 4
あなたの番(1~3): 2
a = 6
computer: 2, a = 8
あなたの番(1~3): 3
a = 11
computer: 1, a = 12
あなたの番(1~3): 3
a = 15
computer: 1, a = 16
あなたの番(1~3): 2
a = 18
computer: 2, a = 20
あなたの番(1~3):
```

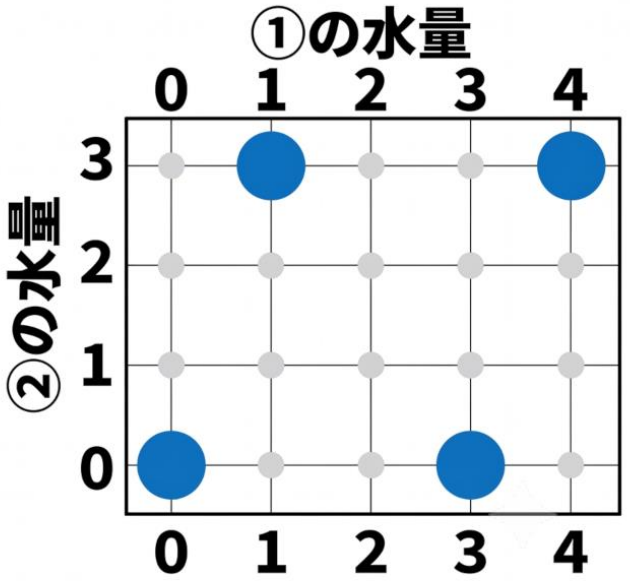
4の倍数

常に4の倍数になっている

演習１．２つの水差し（大きさ４と大きさ３）

8つの行動を2回くり返して、
たどり着ける水の量をすべて見つける

8つの行動×2回



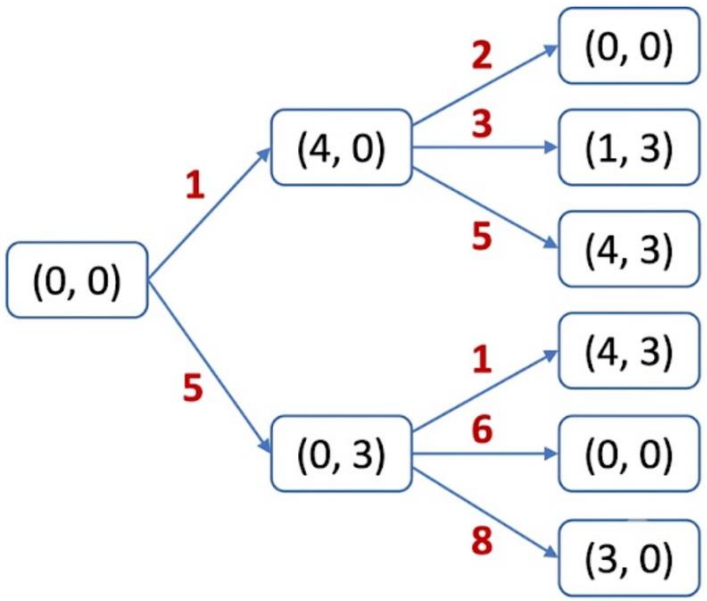
２回の行動の後の
可能性

最初はどちらも0

演習１．２つの水差し（大きさ４と大きさ３）

行動回数２，水差しの大きさ４と３はプログラムで変更可能

行動は２回とする（プログラム内 nstep = 2 で設定）




```
[1,  2] 0 0
[1,  3] 1 3
[1,  5] 4 3
[5,  1] 4 3
[5,  6] 0 0
[5,  8] 3 0
```

プログラムによる総当たりの実行

演習１．プログラムの仕組み

コンピュータに総当たりさせ、2回の行動でたどり着ける水の量をすべて見つける演習

水差し①
容量4

水差し②
容量3

状態（x, y）

x＝①の水量, y＝②の水量

行動：①を満杯／①を空に
①→②へ注ぐ（2通り）
②を満杯／②を空に
②→①へ注ぐ（2通り）

8つの行動を2回くり返す

↓

すべての組み合わせを試す（総当たり）

↓

到達できた水の量を記録

↓

[5, 8, 5, 7] 4 2

行動5→8→5→7 の順で行うと
x=4, y=2 ⇒ 水差し②は2

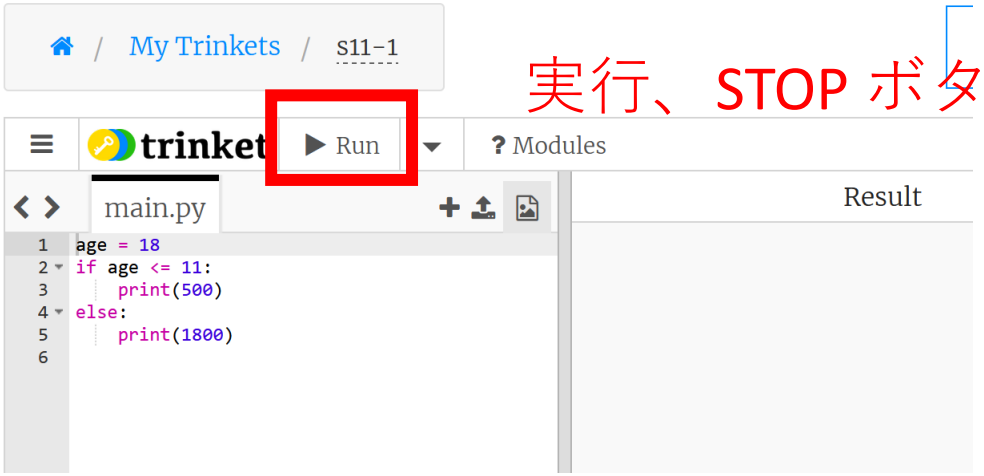
nsteps = 2（行動回数）
※ 3, 4 に変えて再実行できる

繰り返し	すべての経路を試す（総当たり）	for
条件分岐	遷移関数と適用条件	if
出力	経路 最終の水量を表示	print

演習１．trinket でのプログラム実行

- **trinket** は **Python, HTML などのプログラムを書き実行できる**サイト
- https://trinket.io/python/1b2d25b99b

のように、違うプログラムには違う URL が割り当てられる



- 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

行動回数３回（nsteps = 3）に書きかえて再実行

```
40            x, y = x + y, 0
41            ok = True
42
43        return x, y, ok
44
45    def generate_paths(nsteps, seq):
46        if nsteps == 1:
47            return [[i] for i in seq]
48        else:
49            return [path + [i] for path in generate_paths(nsteps - 1, seq) for i i
50
51    nsteps = 3
52    seq = [1, 2, 3, 4, 5, 6, 7, 8]
53
54    for path in generate_paths(nsteps, seq):
55        x, y = 0, 0
56        ok = False
57        for r in path:
58            x, y, ok = move(x, y, r)
59            if not ok:
60                break
61        if ok:
62            print("%s %d %d" % (path, ok and x, y))
```

Powered by trinket

```
[1, 2, 1] 4 0
[1, 2, 5] 0 3
[1, 3, 1] 4 3
[1, 3, 2] 0 3
[1, 3, 6] 1 0
[1, 3, 7] 4 0
[1, 3, 8] 4 0
[1, 5, 2] 0 3
[1, 5, 6] 4 0
[5, 1, 2] 0 3
[5, 1, 6] 4 0
[5, 6, 1] 4 0
[5, 6, 5] 0 3
[5, 8, 1] 4 0
[5, 8, 2] 0 0
[5, 8, 3] 0 3
[5, 8, 4] 0 3
[5, 8, 5] 3 3
```

nsteps = 3 に書き換えて再実行してください
（「3」は必ず半角の数値）

行動回数４回（nsteps = 4）に書きかえて再実行


```
40            x, y = x + y, 0
41            ok = True
42
43        return x, y, ok
44
45    def generate_paths(nsteps, seq):
46        if nsteps == 1:
47            return [[i] for i in seq]
48        else:
49            return [path + [i] for path in generate_paths(nsteps - 1, seq) for i i
50
51    nsteps = 4
52    seq = [1, 2, 3, 4, 5, 6, 7, 8]
53
54    for path in generate_paths(nsteps, seq):
55        x, y = 0, 0
56        ok = False
57        for r in path:
58            x, y, ok =
59            if not ok:
60                break
61        if ok:
62            print("%s %
63
```

```
[5, 1, 6, 2] 0 0
[5, 1, 6, 3] 1 3
[5, 1, 6, 5] 4 3
[5, 6, 1, 2] 0 0
[5, 6, 1, 3] 1 3
[5, 6, 1, 5] 4 3
[5, 6, 5, 1] 4 3
[5, 6, 5, 6] 0 0
[5, 6, 5, 8] 3 0
[5, 8, 1, 2] 0 0
[5, 8, 1, 3] 1 3
[5, 8, 1, 5] 4 3
[5, 8, 2, 1] 4 0
[5, 8, 2, 5] 0 3
[5, 8, 3, 1] 4 3
[5, 8, 3, 6] 0 0
[5, 8, 3, 8] 3 0
[5, 8, 4, 1] 4 3
[5, 8, 4, 6] 0 0
[5, 8, 4, 8] 3 0
[5, 8, 5, 1] 4 3
[5, 8, 5, 2] 0 3
[5, 8, 5, 6] 3 0
[5, 8, 5, 7] 4 2
```

(5, 8, 5, 7) 4 2
行動 **5**, 行動 **8** , 行動 **5**,
行動 **7** の順で行うと
x = **4**, y = **2** になる
⇒ **水差し②は 2 になる.**

nsteps = 4 に書き換えて再実行してください
（「4」は必ず半角の数値）

演習１．ソースコード


```
# 容量4と容量3の2つの水差しのシミュレーション。
# x: 容量4の水差し(①)の水の量、y: 容量3の水差し(②)の水の量。
# r: 行動の番号(1～8)。表の遷移関数・適用条件に従う。

CX = 4  # ①の容量
CY = 3  # ②の容量


def move(x, y, r):
    ok = False
    # 行動1: ①を満杯  x->4   (x < 4)
    if r == 1 and x < CX:
        x = CX
        ok = True
    # 行動2: ①を空に  x->0   (x > 0)
    if r == 2 and x > 0:
        x = 0
        ok = True
    # 行動3: ①で②を満杯  x->x+y-3, y->3   (x+y >= 3 かつ y < 3)
    if r == 3 and x + y >= CY and y < CY:
        x, y = x + y - CY, CY
        ok = True
    # 行動4: ①を全部②へ  y->x+y, x->0   (x+y <= 3 かつ x > 0)
    if r == 4 and x + y <= CY and x > 0:
        x, y = 0, x + y
        ok = True
    # 行動5: ②を満杯  y->3   (y < 3)
    if r == 5 and y < CY:
        y = CY
        ok = True
    # 行動6: ②を空  y->0   (y > 0)
    if r == 6 and y > 0:
        y = 0          # 修正: 元コードの x = 0 はバグ
        ok = True
    # 行動7: ②で①を満杯に  x->4, y->x+y-4   (x+y >= 4 かつ x < 4)
    if r == 7 and x + y >= CX and x < CX:
        x, y = CX, x + y - CX
        ok = True
    # 行動8: ②を全部①へ  x->x+y, y->0   (x+y <= 4 かつ y > 0)
    if r == 8 and x + y <= CX and y > 0:
        x, y = x + y, 0
        ok = True

    return x, y, ok


def generate_paths(nsteps, seq):
    if nsteps == 1:
        return [[i] for i in seq]
    else:
        return [path + [i] for path in generate_paths(nsteps - 1, seq) for i in seq]


nsteps = 2
seq = [1, 2, 3, 4, 5, 6, 7, 8]

for path in generate_paths(nsteps, seq):
    x, y = 0, 0
    ok = False
    for r in path:
        x, y, ok = move(x, y, r)
        if not ok:
            break
    if ok:
        print("%s %d %d" % (path, ok and x, y))
```

演習２．２１ゲーム

「21」ゲーム：後手のコンピュータが必ず勝つ仕組み

ユーザー入力

1 2 3

1、2、3のいずれか

コンピュータの戦略

(4 - a % 4) % 4

4の倍数を作る

4 8 12 16 20

常に4の倍数

最終結果

ユーザーは必ず21に到達し、負ける

21

負け

後手必勝アルゴリズムの実行

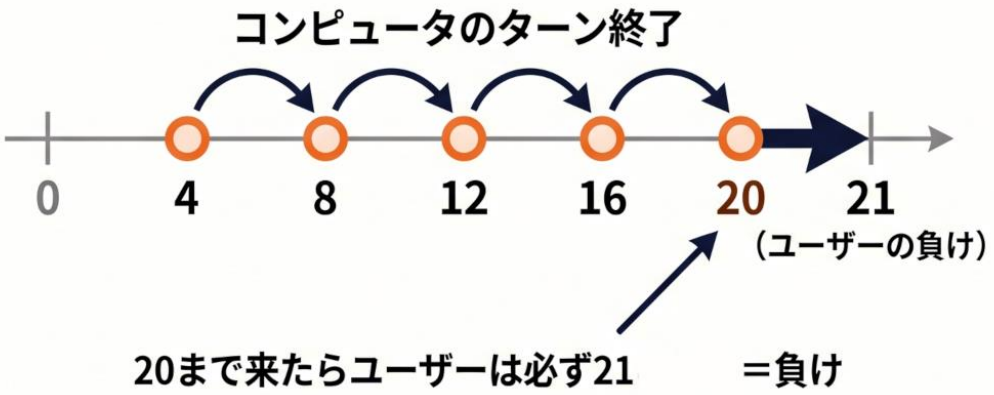
演習２．プログラムの仕組み

後攻のコンピュータは、自分のターン終了時の数を必ず4の倍数にしてユーザーを21へ追い込み、必ず勝つ

必勝のしくみ

ユーザーは１・２・３のいずれかを足す。
21 に到達した側が負け。

「r = (4 - a % 4) % 4」(rが0のときはr=1に補正)



5つの基本動作とコードの対応

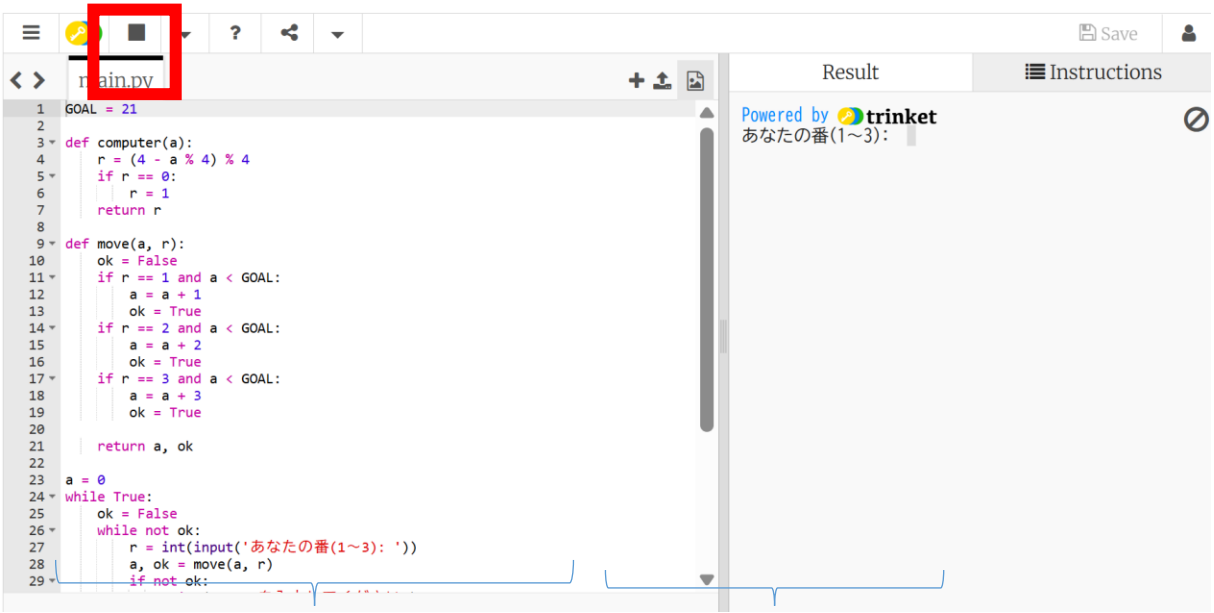
入力	ユーザーが１・２・３を打ち込む	→ input
出力	今の数 a や行動番号を表示	→ print
条件分岐	「数<21 のときだけ可能」 「r==0 のとき r=1」	→ if
繰り返し	ユーザーとコンピュータが交互に行動	→ while
順次実行	ユーザーの行動 → コンピュータの行動 の順に実行	

演習２． 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



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

演習２．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　コンピュータが対戦相手である。画面を確認する。２１になったらゲームは終わり。「**Stop**」をクリックして**終了**。

```
GOAL = 21

def computer(a):
    r = (4 - a % 4) % 4
    if r == 0:
        r = 1
    return r

def move(a, r):
    ok = False
    if r == 1 and a < GOAL:
        a = a + 1
        ok = True
    if r == 2 and a < GOAL:
        a = a + 2
        ok = True
    if r == 3 and a < GOAL:
        a = a + 3
        ok = True

    return a, ok

a = 0
while True:
    ok = False
    while not ok:
        r = int(input('あなたの番(1～3): '))
        a, ok = move(a, r)
        if not ok:
```

```
Powered by trinket
あなたの番(1～3): 3
a = 3
computer: 1, a = 4
あなたの番(1～3): 1
a = 5
computer: 3, a = 8
あなたの番(1～3): 3
a = 11
computer: 1, a = 12
あなたの番(1～3): 2
a = 14
computer: 2, a = 16
あなたの番(1～3): 2
a = 18
computer: 2, a = 20
あなたの番(1～3): 1
a = 21
あなたの負けです
```

マウスでこちら側の画面をクリックし、半角の数字を入れる。入れ間違ったときはDELキー、BSキー

演習２．ソースコード


```
GOAL = 21

def computer(a):
    r = (4 - a % 4) % 4
    if r == 0:
        r = 1
    return r

def move(a, r):
    ok = False
    if r == 1 and a < GOAL:
        a = a + 1
        ok = True
    if r == 2 and a < GOAL:
        a = a + 2
        ok = True
    if r == 3 and a < GOAL:
        a = a + 3
        ok = True

    return a, ok

a = 0
while True:
    ok = False
    while not ok:
        r = int(input('あなたの番(1～3): '))
        a, ok = move(a, r)
        if not ok:
            print('1～3 を入力してください')
    print('a = %d' % a)
    if a == GOAL:
        print('あなたの負けです')
        break

    r = computer(a)
    a, ok = move(a, r)
    print('computer: %d, a = %d' % (r, a))
    if a == GOAL:
        print('コンピュータの負けです')
        break
```