cs-9. Python プログラミング基礎②

（コンピューターサイエンス）

金子邦彦

資料中の図などは Google Nano Banana 2 を用いて作成

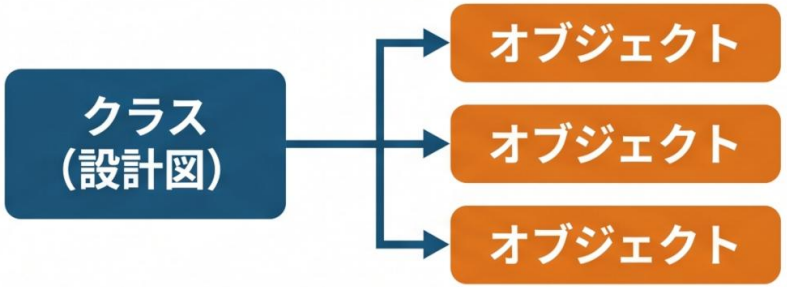
「コンピューターサイエンス」第９回の内容

Pythonの基本：ライブラリ/クラス/条件分岐/リスト/繰り返し

ライブラリ

標準も外部も、使うには import が必要

クラス＝設計図



同じ種類のオブジェクトを作る元

条件分岐（if / else）

```
if 点数 >= 60:
    print("合格")
else:
    print("不合格")
```

点数=80 → 合格

点数=50 → 不合格

条件によって処理を分ける

リスト

0	1	2	3
"赤"	"青"	"緑"	"黄"

リスト[0] → "赤"

複数の要素を順番に保持／
番号は0から始まる

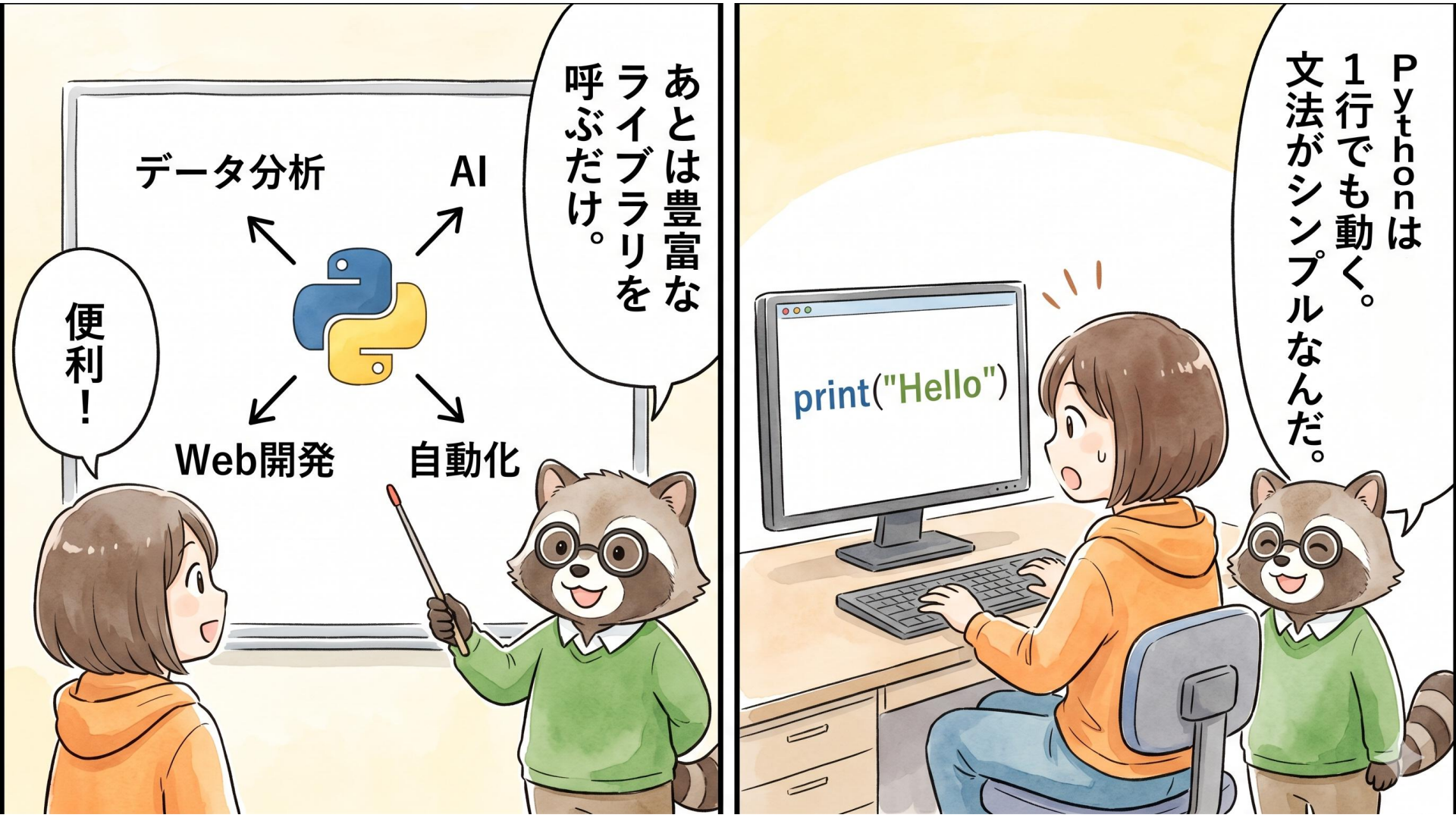
繰り返し（for）

```
for i in range(3):
    print("実行")
```

決まった回数だけ処理をくり返す

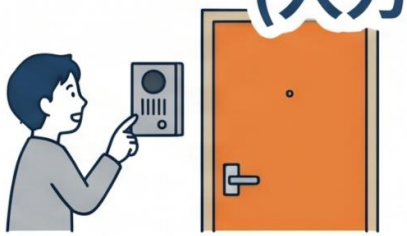
本授業の図は生成AIである Google Nano Banana 2 を用いて作成

9-1. Python の特徴


Pythonは
1行でも動く。
文法がシンプルなんだ。
print("Hello")
あとは豊富な
ライブラリを
呼ぶだけ。
データ分析
AI
Web開発
自動化
便利！

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

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)
```

編集・修正

人間が読める言葉（プログラミング言語）で書かれている

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

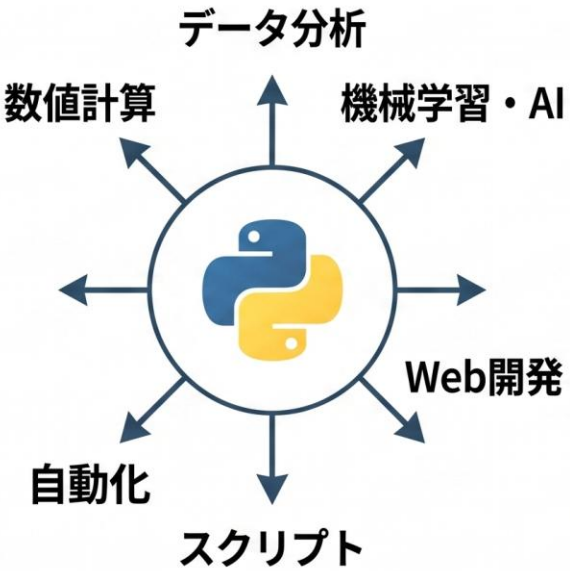
Python には、文法のシンプルさ・ライブラリの豊富さ・柔軟性という利点がある

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

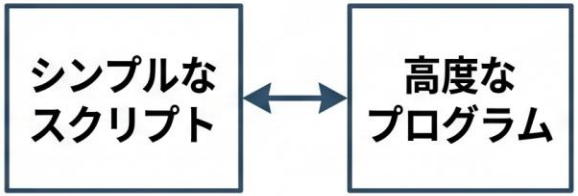
ブロック構造	字下げ（インデント）で示す
	if x > 0:
	　print('正')
	else:
	　print('負以下')

豊富なライブラリ



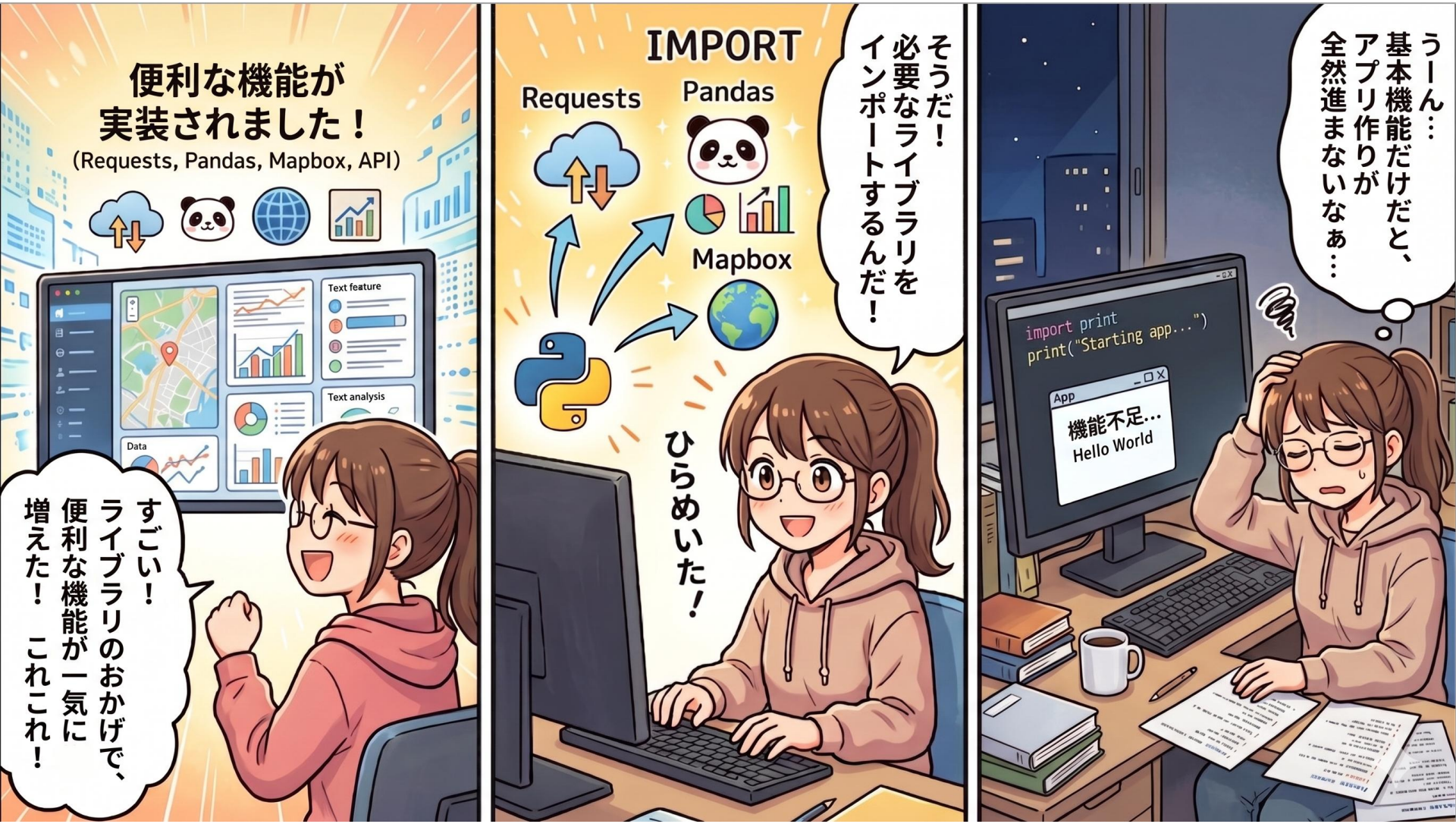
多岐にわたる分野で利用できる

柔軟性



小規模から大規模まで
同じ言語で書ける

9-2. ライブラリとインポート


うーん…
基本機能だけだと、
アプリ作りが
全然進まないなぁ…
import print
print("Starting app...")
App
機能不足…
Hello World
そうだ！
必要なライブラリを
インポートするんだ！
IMPORT
Requests
Pandas
Mapbox
ひらめいた！
便利な機能が
実装されました！
(Requests, Pandas, Mapbox, API)
Text feature
Text analysis
Data
すごい！
ライブラリのおかげで、
便利な機能が一気に
増えた！　これこれ！

Python の代表的なライブラリ

NumPy

数値計算・配列を扱う

1	2	3
4	5	6
7	8	9

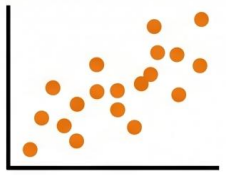
配列

大量の数値をまとめて
高速に計算

外部ライブラリ

Matplotlib

データをグラフにする



```
plt.plot(x, y, 'o')
```

: 'o' は丸いマーカー

折れ線・散布図など
データを目で見える形に

外部ライブラリ

random

予測できない数値を作る

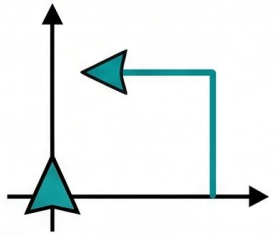
予測不可

シミュレーション /
ゲーム / 暗号化

標準ライブラリ

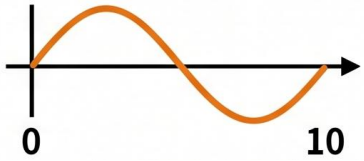
turtle

図形を描いて動かす



原点からスタート、
角度は度(°)で指定

標準ライブラリ



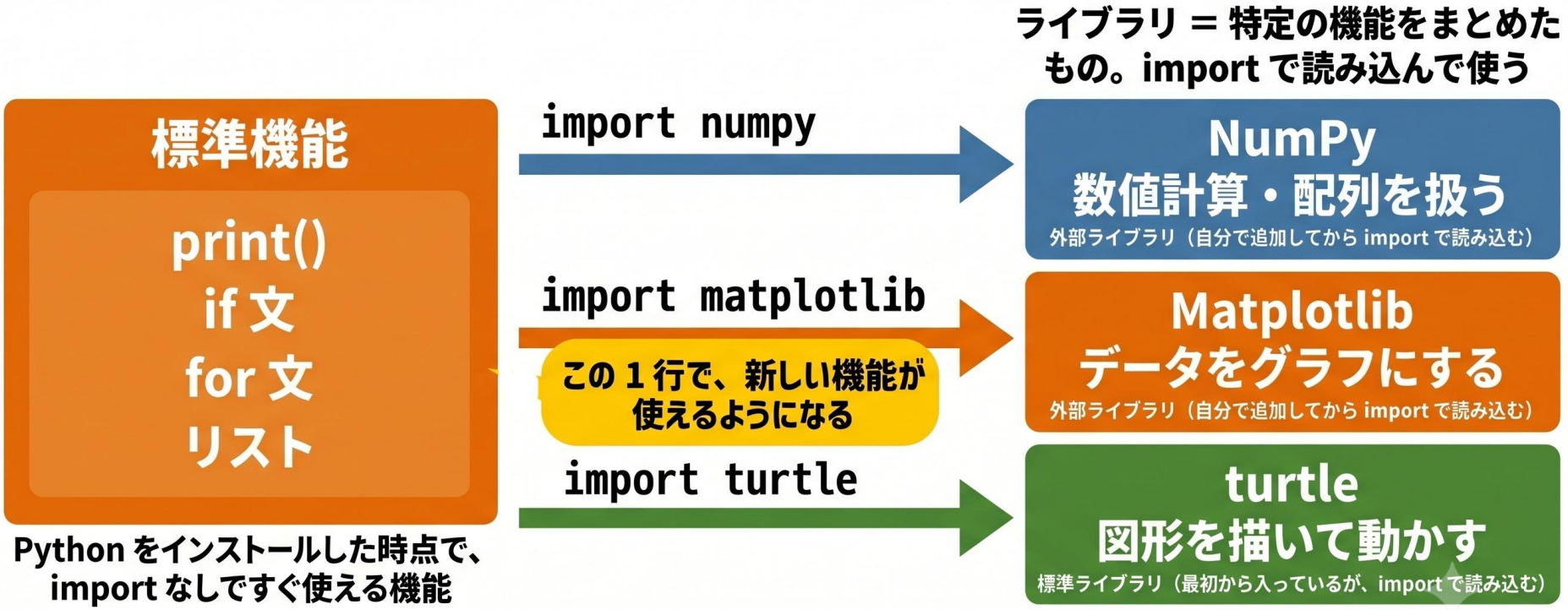
組み合わせるとさらに強力

`x = np.linspace(0, 10, 100)` : 0〜10を100点に分割／点が多いほど滑らか

NumPyで数値を作り、Matplotlibでグラフにする — ライブラリは連携できる

ライブラリを import で読み込む

import って何？ ── 使える機能を増やす１行

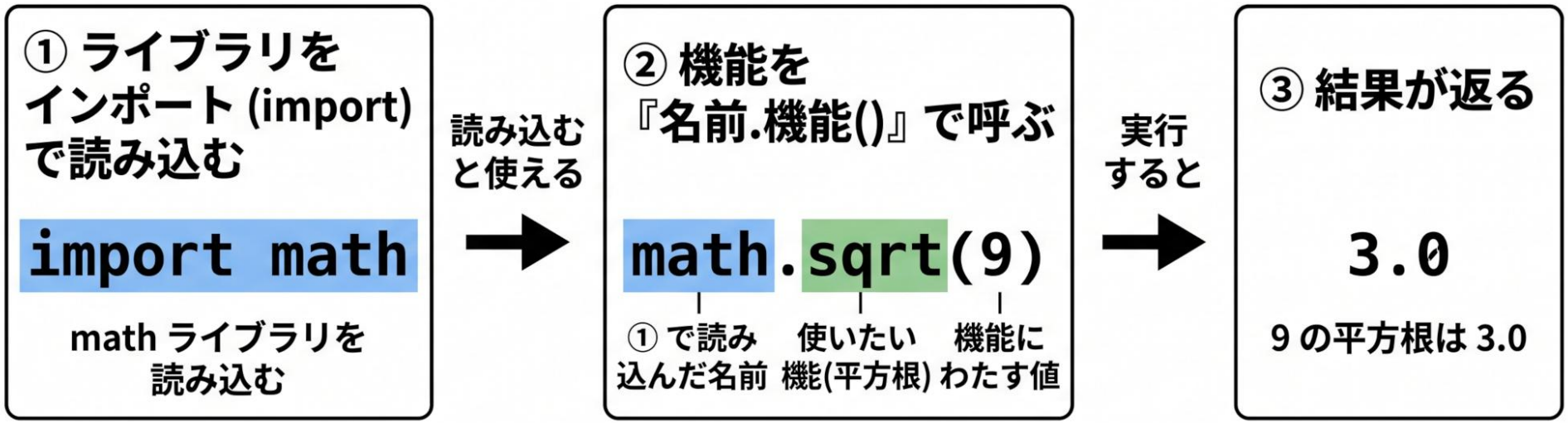


標準機能だけでは足りない処理は、ライブラリを import で読み込んで補う

ライブラリをインポートで読み込む

Database Lab.

例：math ライブラリの sqrt（平方根）を使う



インポート（import）で読み込んだ名前を頭に付けて『名前.機能()』と書くと、その機能が使える

ライブラリのメリット

ライブラリを使うと、少ないコードで高度な機能を効率よく利用できる

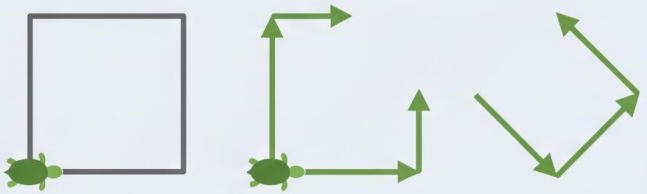
自分で全部作る場合

```
import turtle, 0)                    print(tinteston.sion(1, 31, 17)
print(point(1, 0)                    print(tinteston.sion(2, 31, 19)
print(point(2, 0)                    print(tinatioco)
print(point(3, 0)                    print(tinatioco)
print(roint(5, 15)                   print(tinatiosge)
print(rointees.siint(3,4/5, 10)
print(rointɛ = u(l)                  print(tiotesion.sion(3, 21, 19)
print(rointees.siint(3,1/5, 13)      print(tiɛa)
print(rointɛ = u(l)                  print(linat(39, 42))
print(rointees.siint(5,1/5, 25)      print(tinatiost)
print(rointɛ = u())                  print(tinat(25, 46))
print(rointees.eiint(5,1/5, 55)      print(tintesion.sion(3, 25, 19)
print(rointɛ = u())                  print(linoz) == 1)
print(rointeeas.line(5,1/5, 35)      print(tinat(39, 42))
                                     print(tinoɪ(39, 46))
print(point(1, 0)                    print(tinos(39, 48))
print(point(1, 0)                    print(linoɛ) == 1)
print(point41, 0)                    print(tintesion.ston(3, 20, 46)
print(rointees.stint(5,1/5, 39)      print(tinat) == 1)
print(roints = ʜ())                  print(tinos(49, 12))
print(rointees.stint(3,1/5, 30)      print(tinos(49, 48))
print(rointe = ʜ())                  print(tinatios4)
print(rointees.stint(3,1/5, 30)      print(tinas(93, 36))
print(roints = ʜ())                  print(tiotesion.sten(3, 20, 39)
print(rointees.stint(3,1/5, 30)      print(tinosios4)
print(roint11, 12)                   print(tinatiosit)
print(roint01, 20)
print(roint39, 26)                   print.top(ioun()
print(roint67, 12)                   print(1)
print(roɲntmees.lime(4,01, 12))
print(roɲntmees.lime(4,02, 12))
print(roɲntmees.lime(4,53, 36))
print(roɲntmees.lime(4,54, 34))
print(tonptmees.lime(5,35, 29))
```

コードの量が多い
作成の手間が大きい

ライブラリ（turtle）を使う場合

```
import turtle
turtle = turtle("vvɪd", "sq", 0")
turtle(shapes)
```



短いコードで済む
図形を描いて動かせる
高度な機能をすぐ使える

ライブラリの活用は、ソフトウェア開発に不可欠である

ライブラリのインポート

import の後、機能はこう書いて使う

① そのままの名前で使う

```
import math
print(math.sqrt(7))
```

import した名前を、頭に付けて『math.機能()』と書く

② as で名前を短くできる

```
import numpy as np
x = np.linspace(0, 10, 100)
```

numpy を np という短い名前で呼ぶ約束

毎回 numpy と書くのは長い。as np で短い別名を付けられる

③ 長い名前は as でまとめる

```
import matplotlib.pyplot as plt
plt.plot(x, y, 'o')
```

matplotlib.pyplot は長いので、plt という別名にする

import した名前（または as で付けた別名）を頭に付けて『名前.機能()』と書く

標準ライブラリと外部ライブラリ

Database Lab.

種類ごとに違う便利な機能

標準ライブラリ（インストール不要・最初から使える）

random

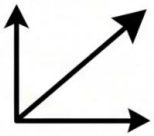
予測できない数値を作る。
ゲームやシミュレーションに

math

$\sqrt{\sqrt{9}} = 3$

平方根・三角関数など
数学の計算ができる

turtle



図形を描いて動かす。
プログラムで絵が描ける

datetime

日付や時刻を扱う。
経過日数の計算などに

os

ファイルやフォルダを
プログラムから操作する

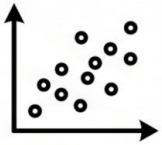
外部ライブラリ（使用には追加（インストール）が必要）

NumPy

1	2	3
4	5	6
7	8	9

大量の数値や配列を
まとめて高速に計算する

Matplotlib



データを折れ線・散布図
などのグラフにする

pandas

表形式のデータを
読み込み・集計・加工する

Pillow

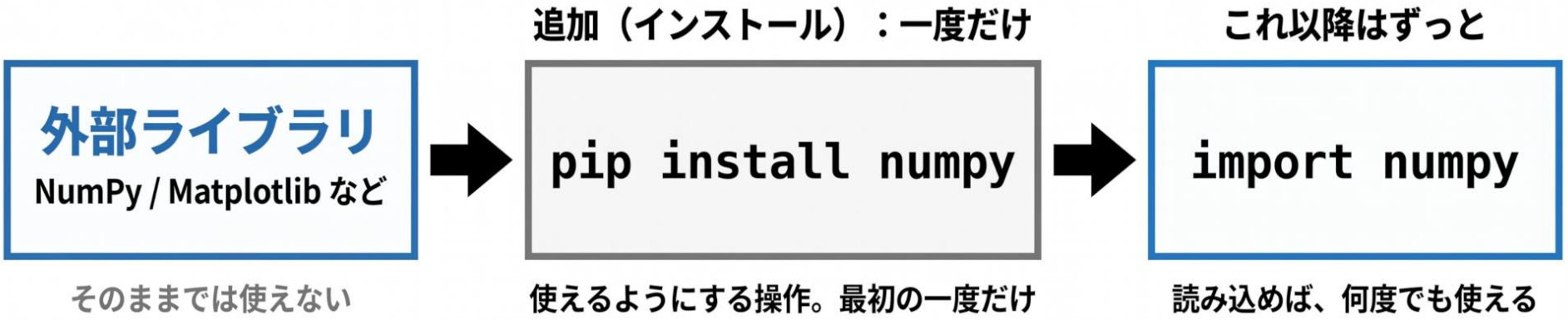
画像を開く・加工する
・保存する

requests

インターネット上の
データを取得する

目的に合ったライブラリを import で読み込めば、いろいろな処理が簡単にできる

外部ライブラリは、使う前に一度だけ追加（インストール）を行う



標準ライブラリ（math / random / turtle など）は、インストール不要。すぐ import で読み込める

※ 学習用の Trinket では、NumPy や Matplotlib などもインストール不要で import できる

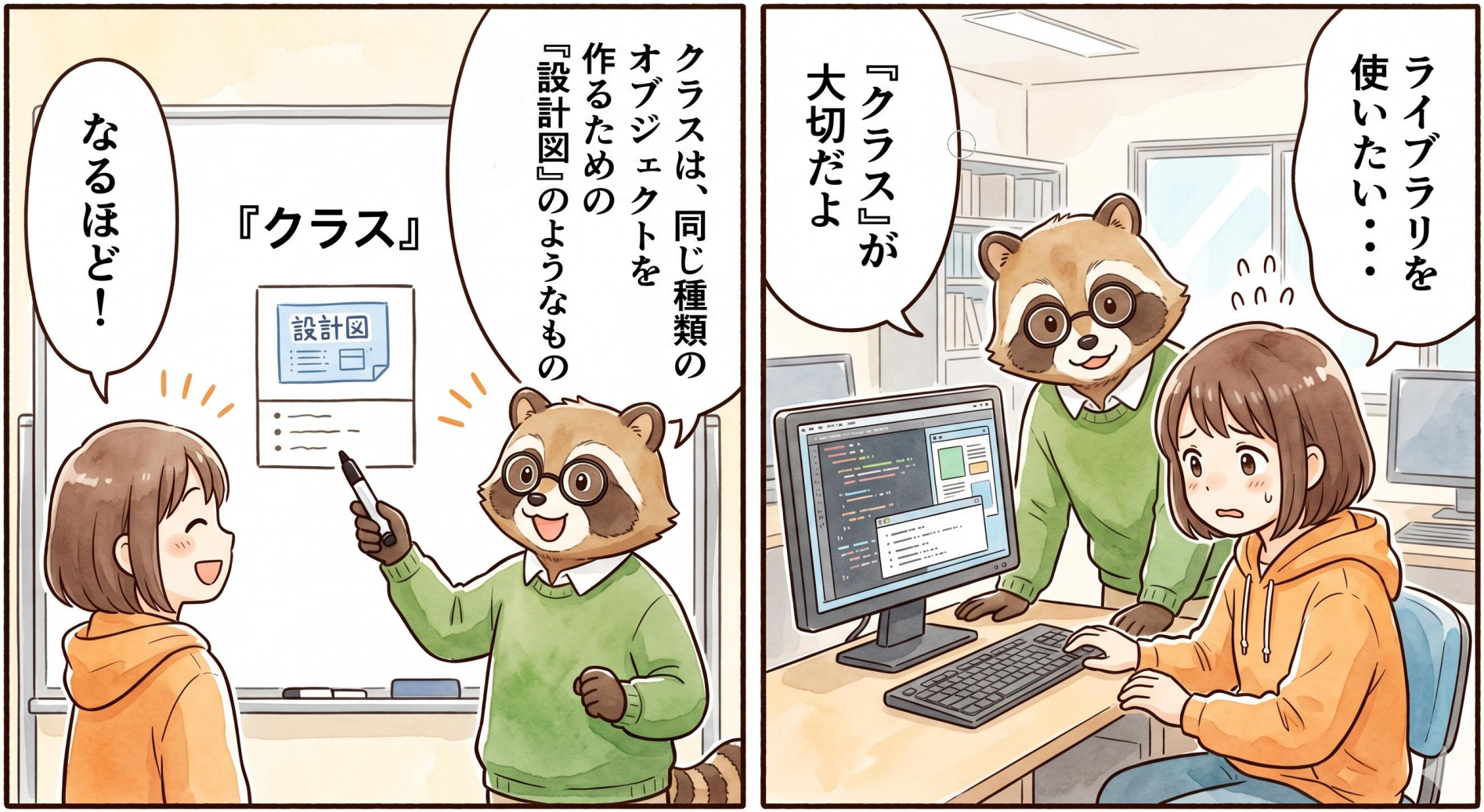
標準ライブラリと外部ライブラリ ── 違いと共通点

項目	標準ライブラリ	外部ライブラリ
インストール	不要（最初から入っている）	必要（本来は pip install が必要）
使うまでの手順	import で読み込むだけ	インストール → import で読み込む
例	math / random / turtle	NumPy / Matplotlib
import	必要	必要

共通点：どちらも import で読み込んでから使う

```
import math
import numpy
```

9-3. オブジェクトとクラス

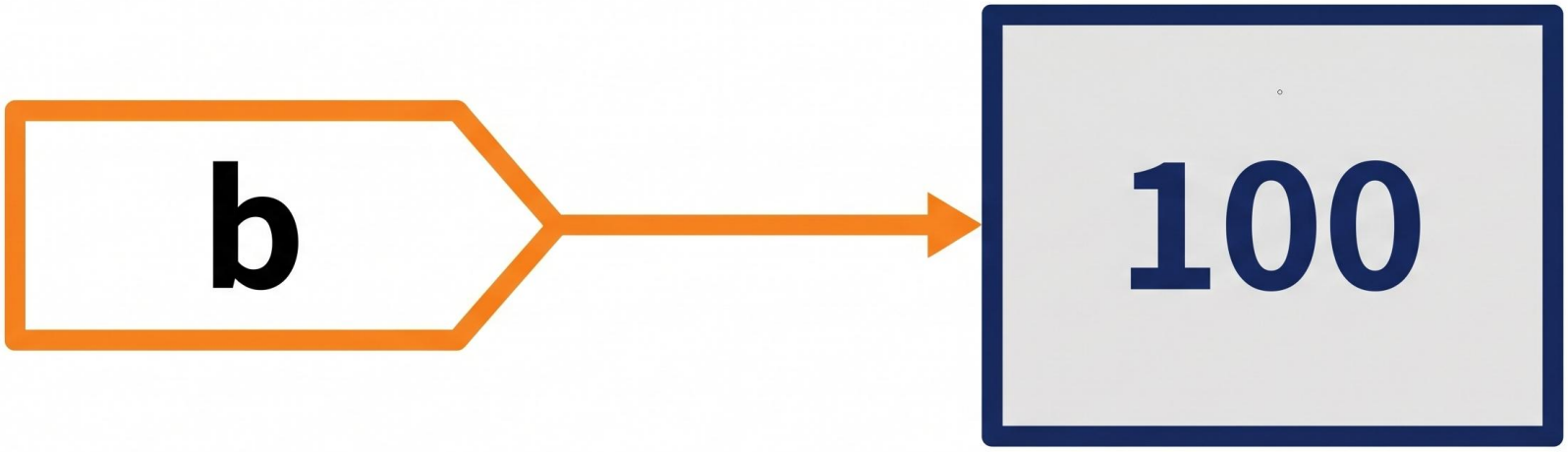

ライブラリを
使いたい・・・
『クラス』が
大切だよ
『クラス』
設計図
クラスは、同じ種類の
オブジェクトを
作るための
『設計図』のようなもの
なるほど！

オブジェクト・メソッド・引数

処理を表す三点セット

変数

変数は『名札』、オブジェクトは『実体』。名札が実体を指し示す。



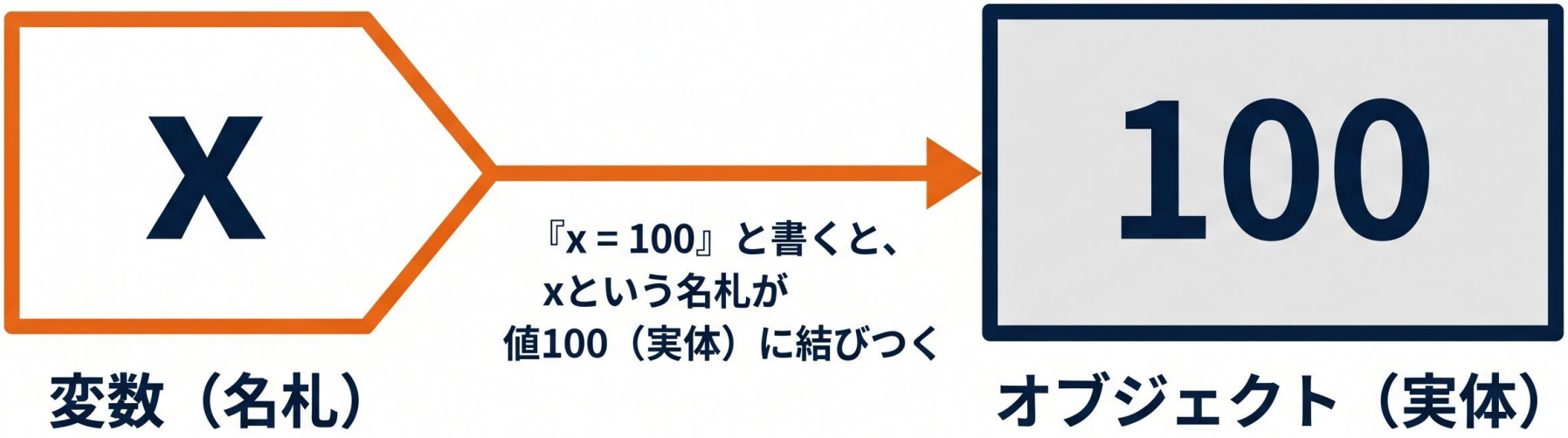
変数（名札）　　**オブジェクト（実体）**

1. データに名前を付けて保管する
2. 変数（名札）は実体を指し示すだけ
3. 必要に応じて書き換えられる

変数と代入

代入とは、名札（変数）に実体（値）を結びつけること

$$x = 100$$



クラスとオブジェクト

1つの型（クラス）から、同じ形のクッキー（オブジェクト）を何枚も作ることができる。

クラスとオブジェクト

クラス：同じ種類のオブジェクトに共通する型を定める

クラス（設計図）

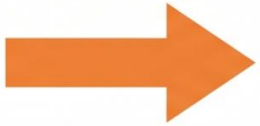
クラス名：自動車

持つ情報：色、速度

できる操作：走る、止まる

同じ種類のオブジェクトに共通する型を定める

この設計図から作る



オブジェクト（実体）

オブジェクト1：色＝赤、速度＝0

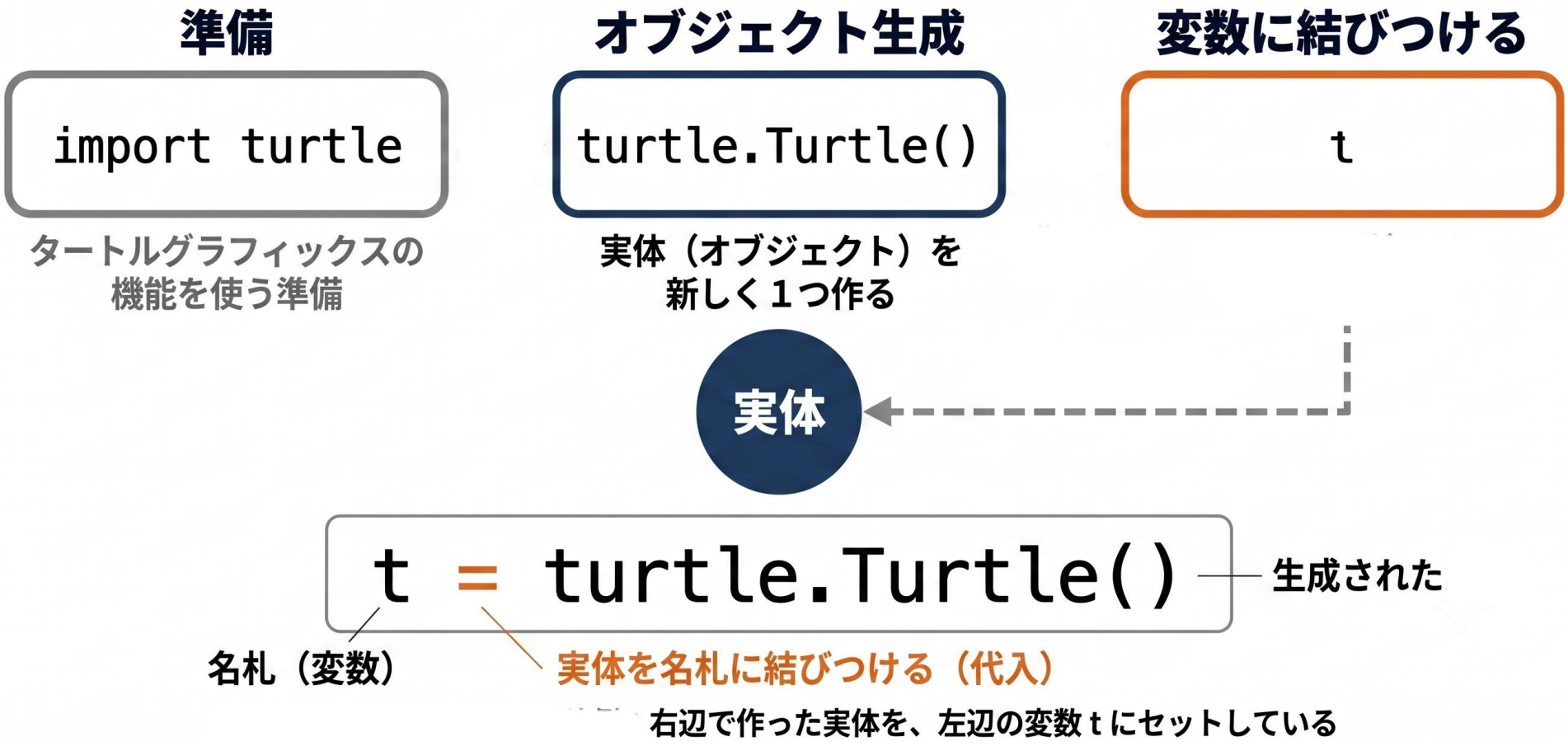
オブジェクト2：色＝白、速度＝60

オブジェクト3：色＝黒、速度＝40

処理の対象となる、個々の実体

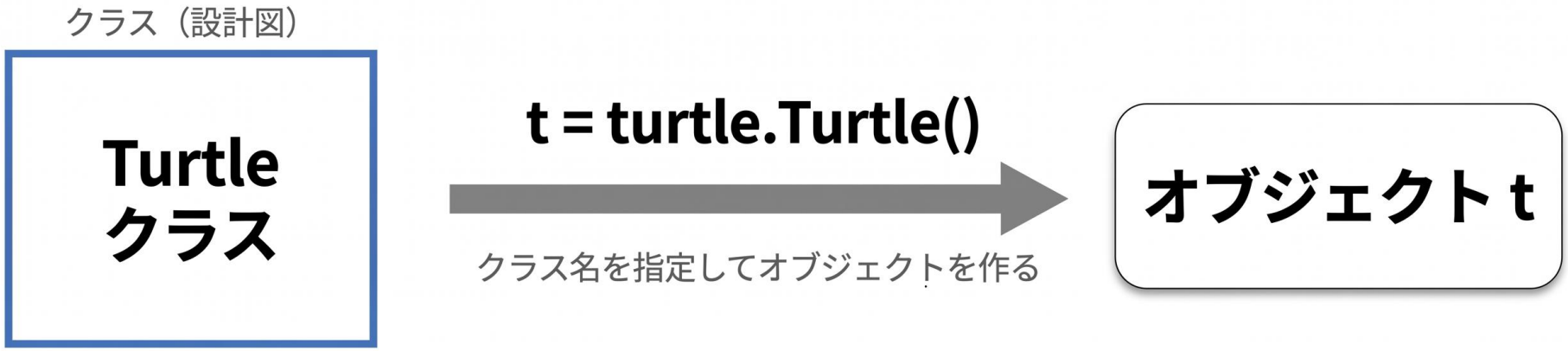
クラスは同じ種類のオブジェクトの設計図であり、同じ種類のオブジェクトの集まりと考えることもできる

オブジェクト生成と代入



クラスとオブジェクト

クラスは設計図、オブジェクトはクラスから作られた実体



オブジェクト.メソッド(引数)

処理の対象　オブジェクトの能力　能力を細かく指定

```
t.goto(0, 100)
```

クラスを使う利点

① 同じクラスから複数のオブジェクトを生成

```
t1 = turtle.Turtle()

t2 = turtle.Turtle()
```

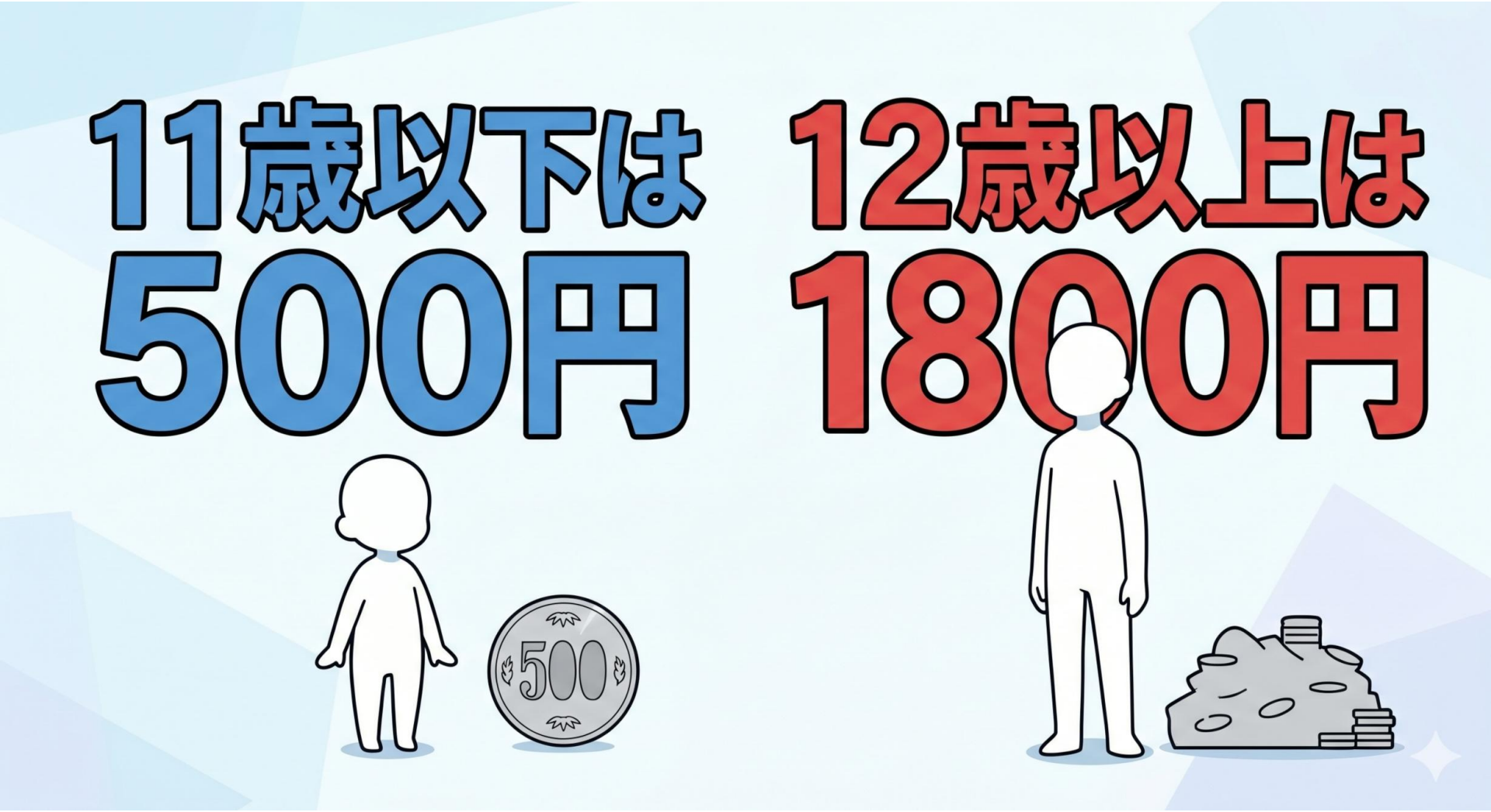
同じ設計図から何個でも作れる

② どちらも同じメソッドで操作できる

同じクラスから作ったオブジェクトは、すべて同じ手順で生成し、同じメソッドで操作できる——この一貫性がクラスの利点

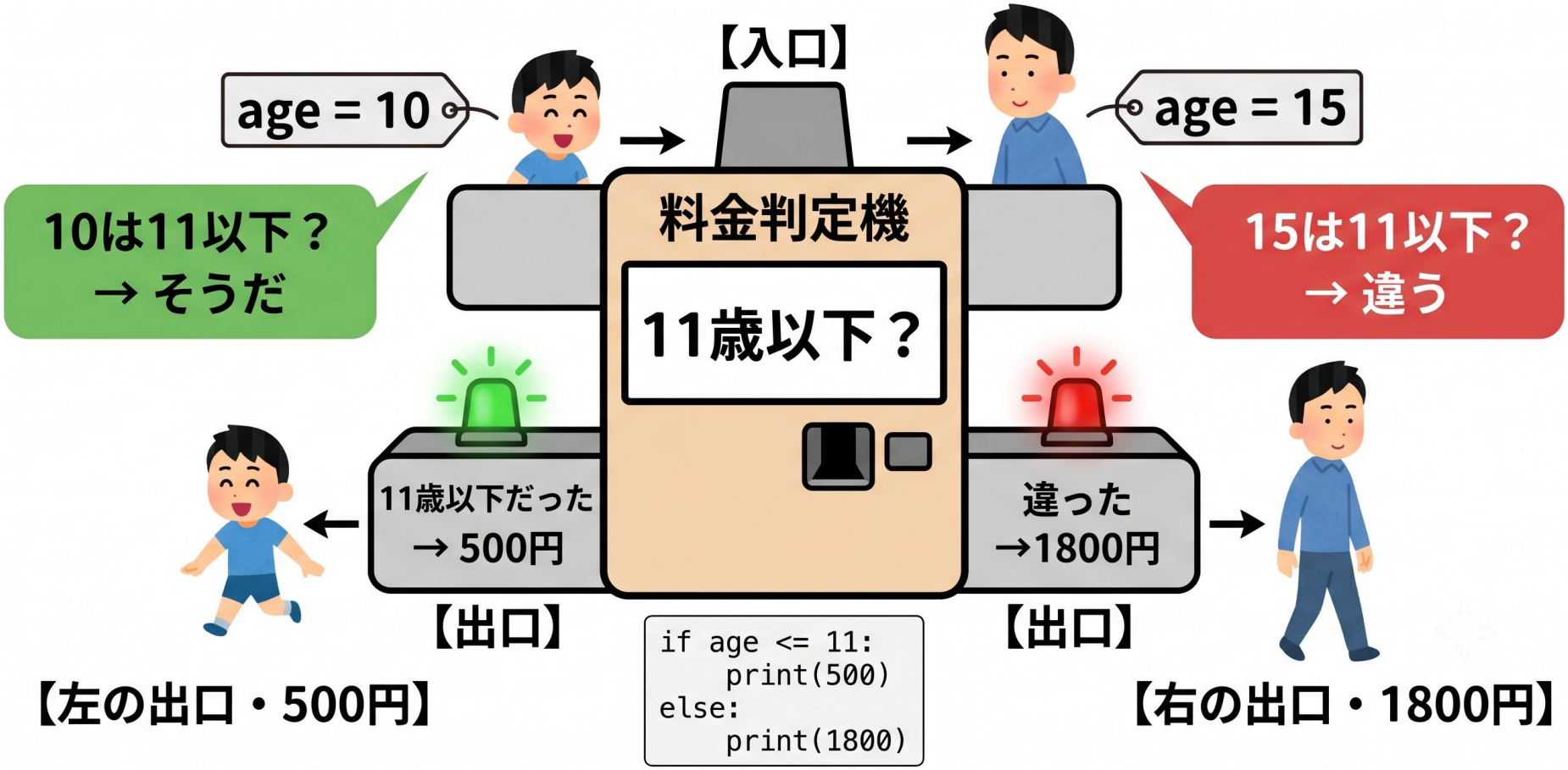
9-4. 条件分岐

条件分岐



条件分岐

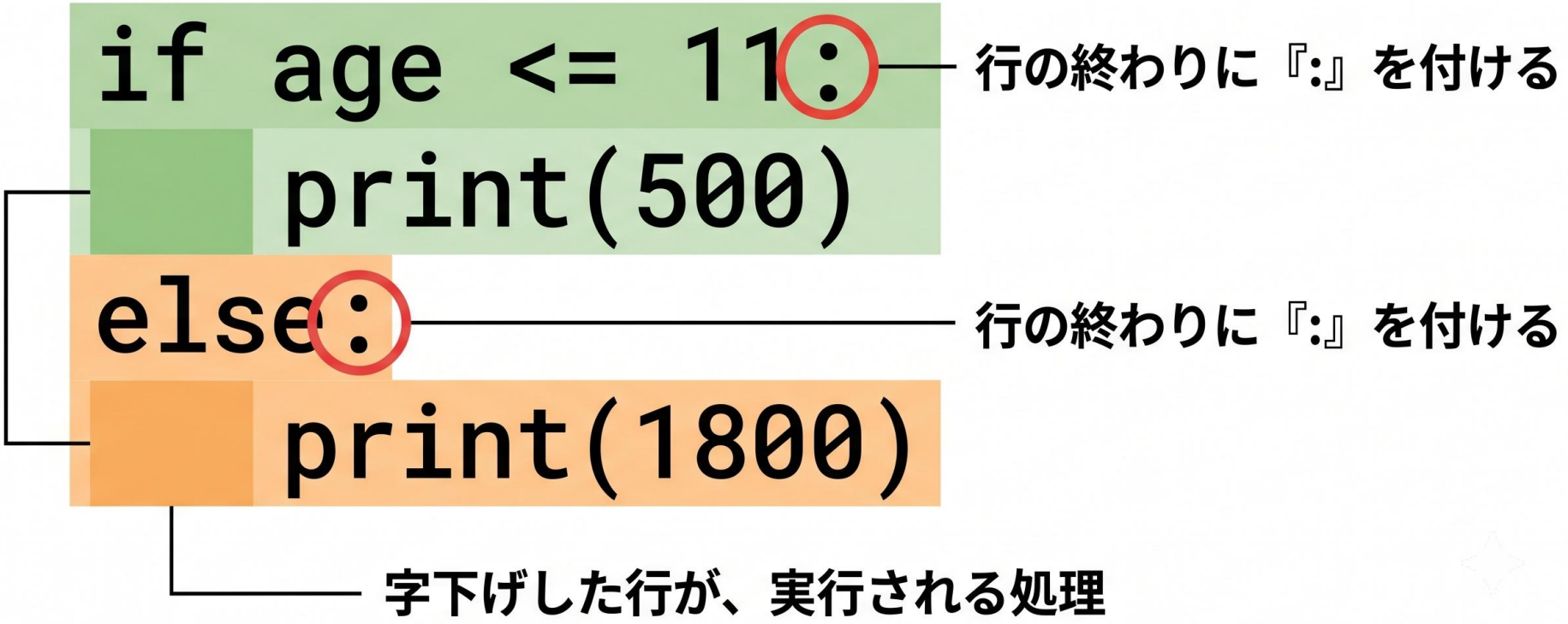
年齢で料金が変わるしくみ




```
if age <= 11:
    print(500)
else:
    print(1800)
```

if 文の書き方

変数の値によって処理を分ける – 年齢で料金を決める例



if 文の書き方

変数の値によって処理を分ける ― 年齢で料金を決める例

条件式はこの1つだけ。
『11以下か？』を確かめる

行の終わりの『:（コロン）』は
『ここから処理が始まる』合図。
書き忘れに注意

```
if age <= 11:
    print(500)
else:
    print(1800)
```

条件が成り立たなかったとき
(11より上)は、自動でこちら側。
だから条件式を２つ書く必要はない

先頭を字下げ（半角スペース）した行が、
その条件 のときに実行される範囲

字下げを間違えると正しく 動かない

age = 10
10 ≦ 11 → あてはまる
→ print(500) が実行される

age = 15
15 ≦ 11 → あてはまらない
→ else 側の print(1800) が実行される

9-5. リスト

リスト

リスト

リスト：複数の要素を順番に保持
番号（インデックス）は0から始まる

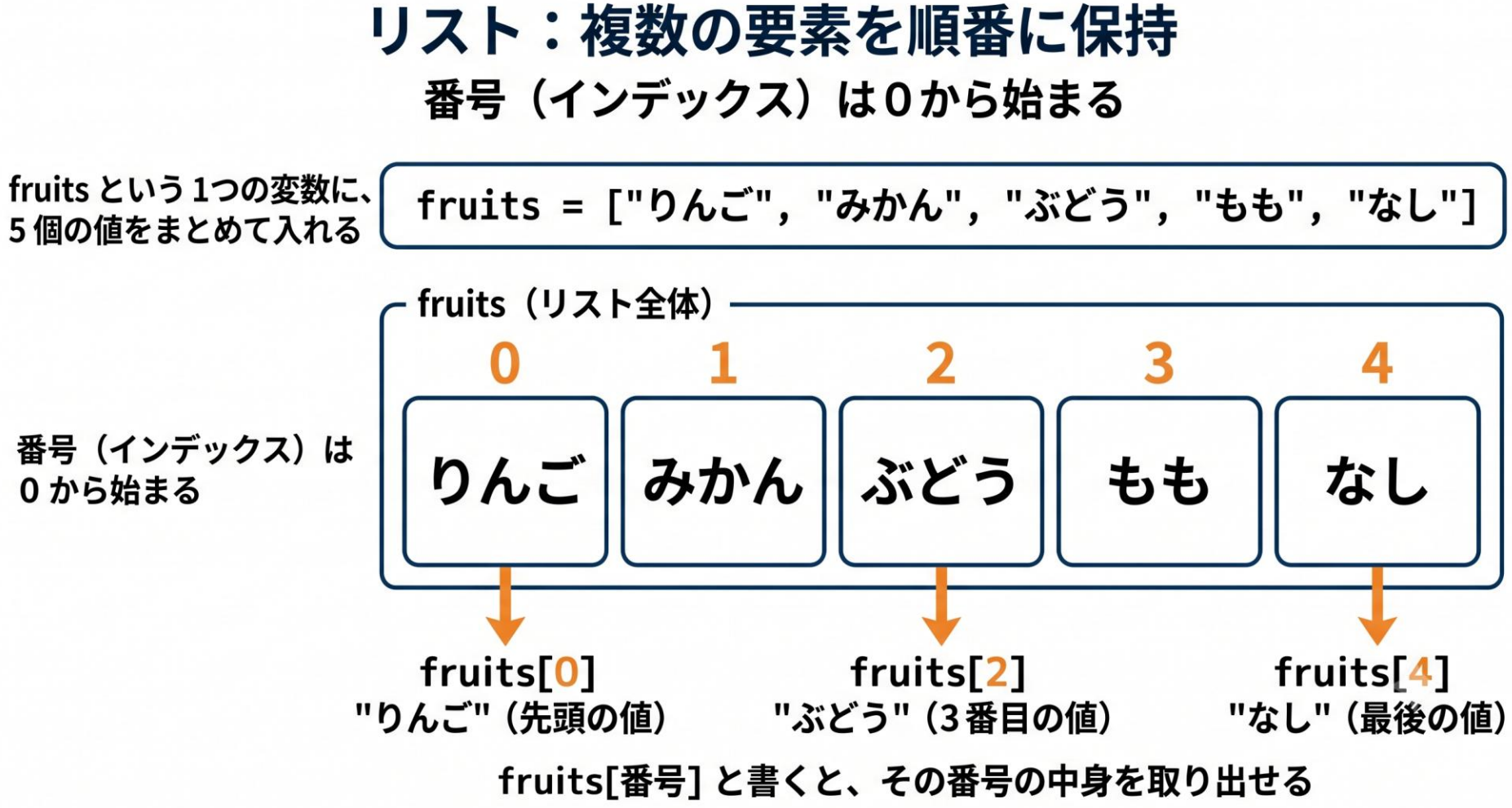
fruits（リスト全体）　fruitsという1つの変数に、5個の値をまとめて入れる

0	1	2	3	4
				
りんご	みかん	ぶどう	もも	なし

fruits[0] →“りんご”（先頭の値）　fruits[2] →“ぶどう”（3番目の値）　fruits[4] →“なし”（最後の値）

fruits[番号] と書くと、その番号の中身を取り出せる

リストを扱う Python プログラム

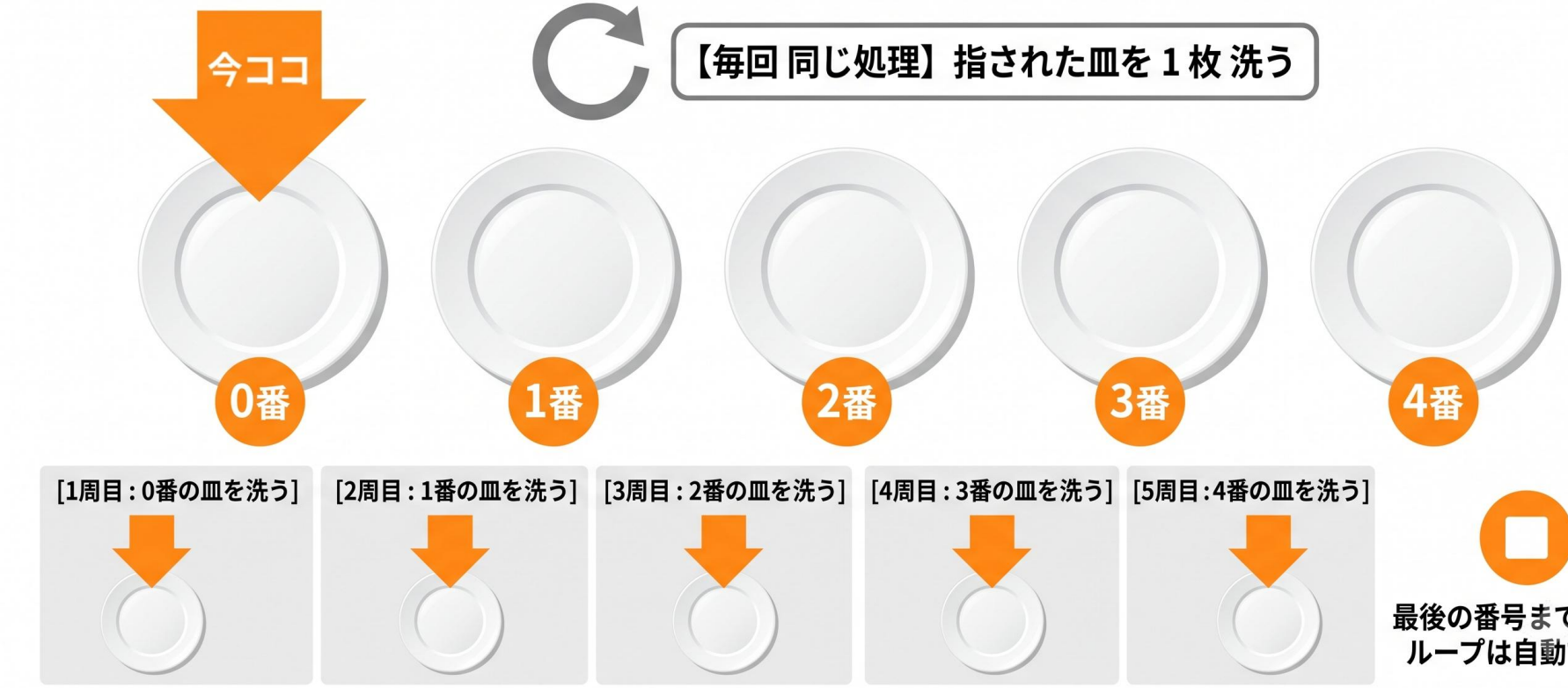


9-6. ループ（繰り返し）

ループ(繰り返し）：同じ処理を繰り返す

ループ(繰り返し）：同じ処理を繰り返す

ループ　並んだ対象を、先頭から番号順に1つずつ、同じ処理をしていく場合



変わらないもの ＝ 処理の中身（毎回“皿を1枚洗う”で同じ）

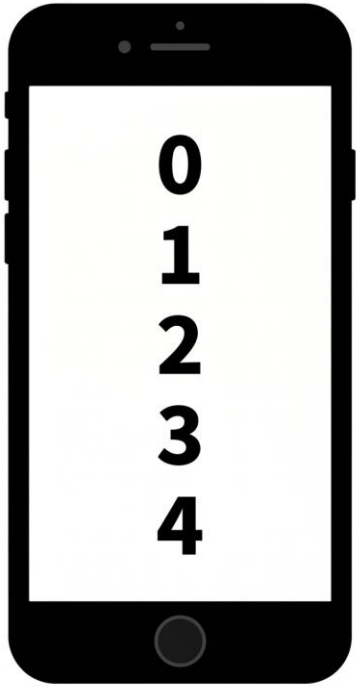
変わるもの ＝ 対象の番号（0→1→2→3→4 と1つずつ進む）

ループ（繰り返し）

Python の for によるループ（繰り返し）

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

プログラム



実行結果

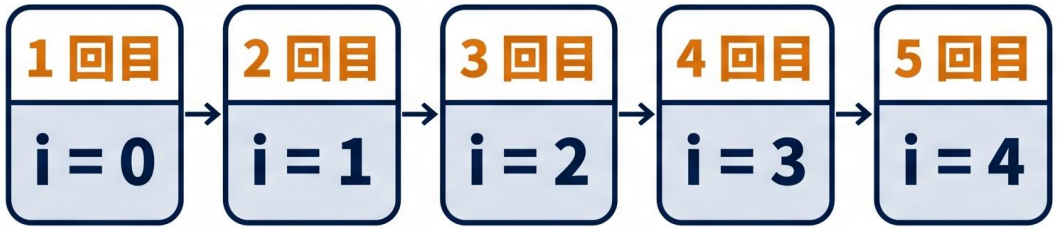
- 「range(5)」は，**0 から 4 までの 5個の数**
- 「**for i in range(5)**」の場合，**i の値**は，**0, 1, 2, 3, 4 と変化**
- 「print(i)」の部分は，それぞれ i の値に対して 1 回実行

Python の range

range(5) は 0 から 4 までの 5 個の数。回数と i の値は別物

range(5) の中身

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



回数は 1 から、i は 0 から始まる

5 個の数 → 0・1・2・3・4 (5 は入らない)

i は計算に使える

```
for i in range(5):
    print(i * 2)
```

i の値		i * 2 の出力
0	→	0
1	→	2
2	→	4
3	→	6
4	→	8

i = 0 のとき出力も 0 になる点に注意

Python の for の書き方

for

同じ処理を繰り返す。字下げした部分が繰り返される。

Python

行末のコロンを忘れない

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

← この字下げした部分が繰り返される

字下げ（インデント）は半角スペース4つが基本
字下げをやめると、繰り返しの範囲から外れる

1回目：i = 0 → print(i) 実行

2回目：i = 1 → print(i) 実行

3回目：i = 2 → print(i) 実行

4回目：i = 3 → print(i) 実行

5回目：i = 4 → print(i) 実行

繰り返し終了

実行結果

```
0
1
2
3
4
```

リスト

Python の for によるループ（繰り返し）

```
for fruit in ["りんご","みかん","ぶどう","もも","なし"]:
    print(fruit)
```

プログラム

実行結果

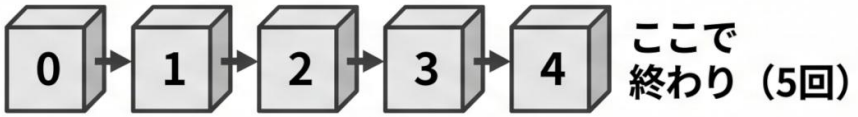
for と while の使い分け

Database Lab.

繰り返す回数が決まっていれば for、条件しだいで終わるなら while

for ＝ 回数が決まった繰り返し

繰り返す対象や回数が、
始める前からわかっているとき




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

箱の数だけ繰り返すので、
回数は最初から確定している

while ＝ 条件が続く間の繰り返し

いつ終わるかは、
やってみないとわからないとき


```
while 残高 > 0:
    引き出す()
```

回数ではなく『残高が0より大きい間』という
条件で続く。何回で終わるかは事前にわからない

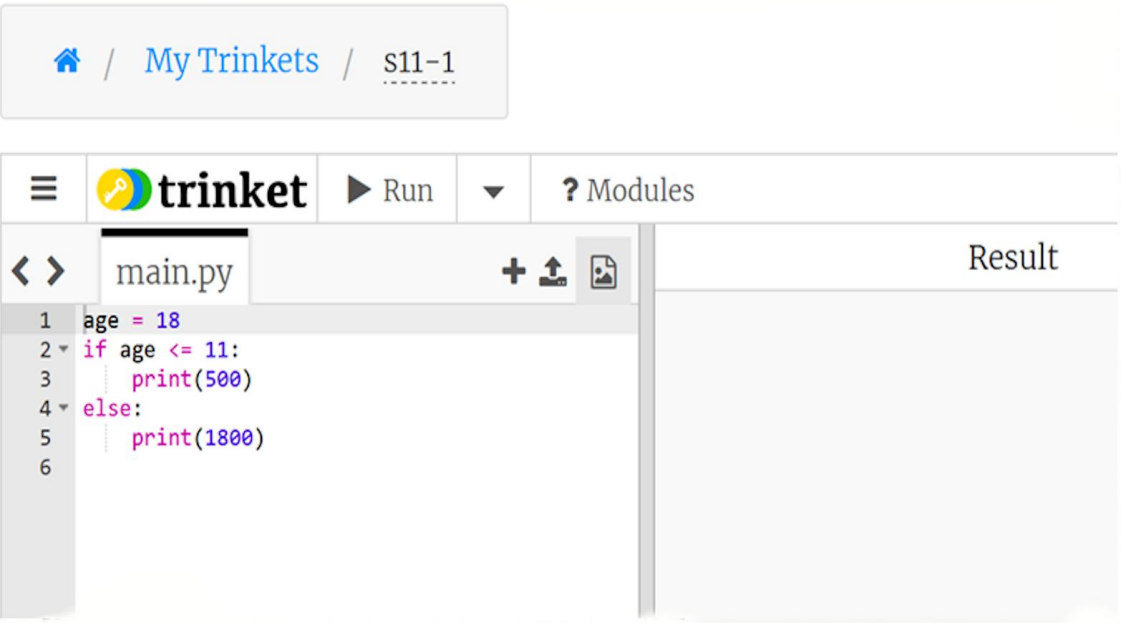
繰り返す回数は、始める前から決まっている？

決まっている	決まっていない
for	while

演習

Trinket 操作手順

Trinket は『Run』で実行し『STOP』で停止、左で編集・右で結果確認



① ここで編集して再実行できる

② 結果はここで確認する

結果表示について

実行環境によってはこう表示される

('x =', 200)

⇒

次のように読めばよい

x = 200

括弧とカンマは表示の違い。間違いではない

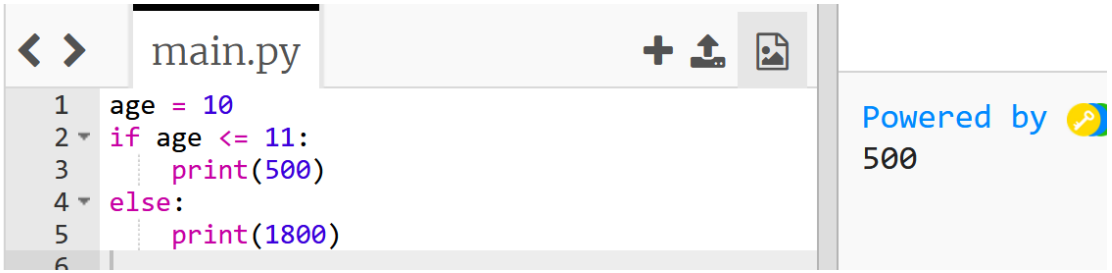
演習１．条件分岐

①trinketの次のページを開く

https://trinket.io/python/0fd59392c8

② 実行する．1800が表示されることを確認

③ 「age =**18**」を「age =**10**」に書き替える



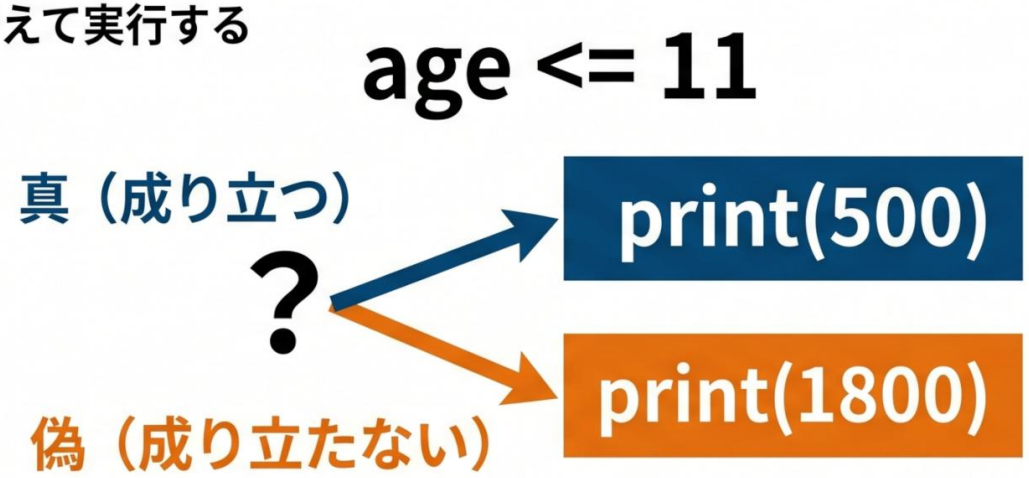
④ 実行する．**500が表示される**ことを確認

⑤ageの値が8, 9, 10, 11のときは500になり，
12, 13, 14, 15のときは1800になることを確認

age の値によって出力が 500 か 1800 に分かれる

```
age = 18
if age <= 11:
    print(500)
else:
    print(1800)
```

ここを書き替えて実行する



演習２．条件分岐

①trinketの次のページを開く

https://trinket.io/python/62f74d3bfc

② 実行する.

【課題の整理】

何を作るか

- weight が 100 以下 → 0
- weight が 100 より大きい → 1000

テスト条件：weight = 80

【プログラムと実行結果】

プログラム

```
weight = 80
if weight <= 100:
    print(0)
else:
    print(1000)
```

出力：0

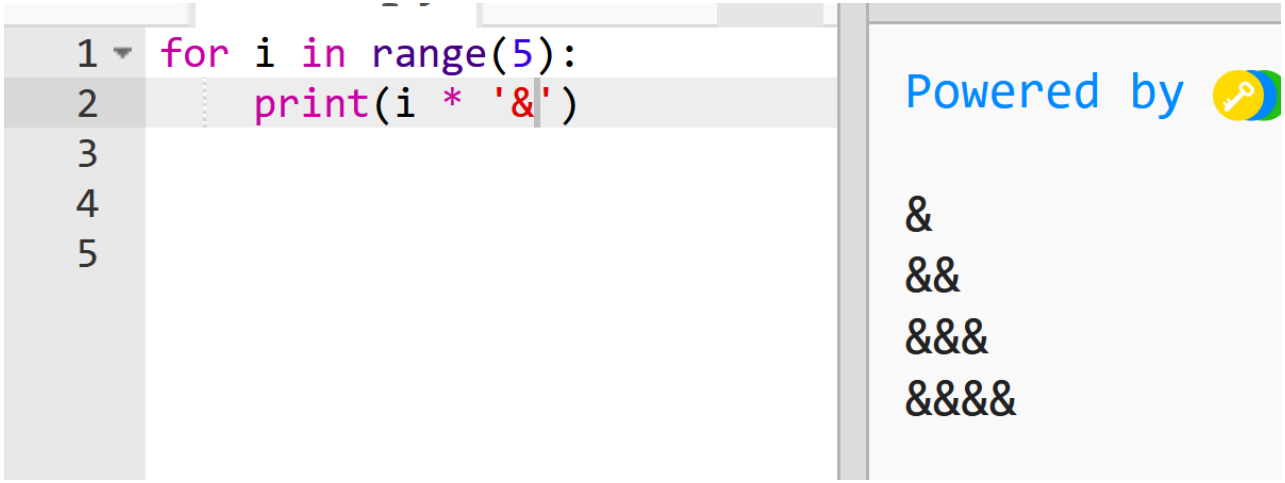
条件 weight <= 100 が成り立つかどうかで、実行される行が分かれる。今回は 80 なので 0 が出力される。

演習３．for による繰り返し

①trinketの次のページを開く

https://trinket.io/python/27f6ebe1da

② 実行結果が，次のように表示されることを確認



- i=0 のときは &が0個（空行）になる
- 数値どうしの * は掛け算だが、「文字列 * 数」 はその文字列を繰り返す。「3 * '&'」 は 「'&&&'」

③ 確認クイズ

&を9個まで表示したい（下図のように）ときは,
どのようにプログラムを書き換えるか？
自分でチャレンジしてせよ

```
1 for i in range(5):
2     print(i * '&')
```

このプログラムを書き換える

```
&
&&
&&&
&&&&
&&&&&
&&&&&&
&&&&&&&
&&&&&&&&
&&&&&&&&&
```

このような結果を得る

ヒント：range(10) に変えると i=9 で
&が9個になる（先頭に空行が出る）

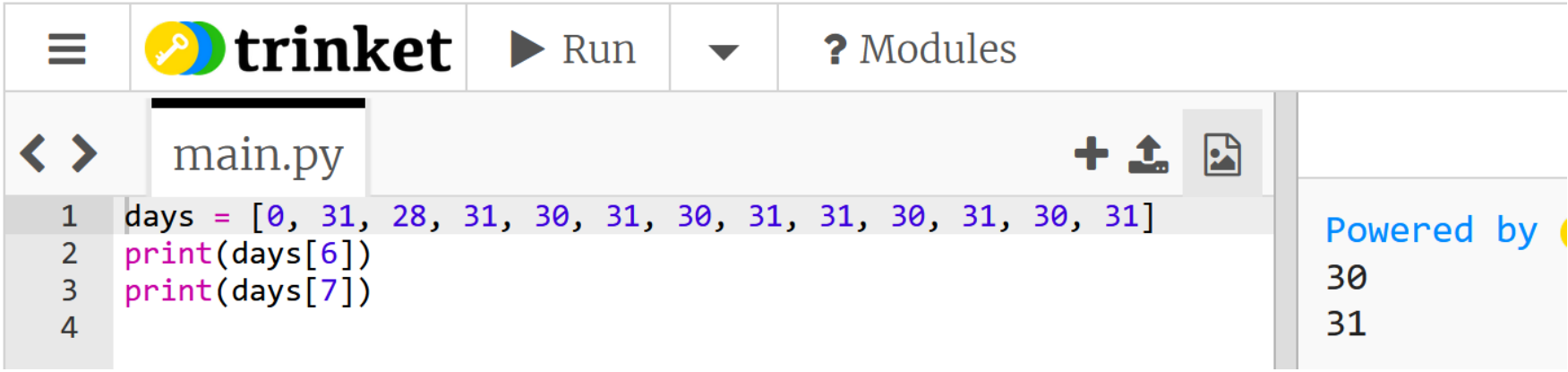
演習４．月の日数

6月は**30**日まである．**7**月は**31**日まである．
※うるう年のことは考えないことにする

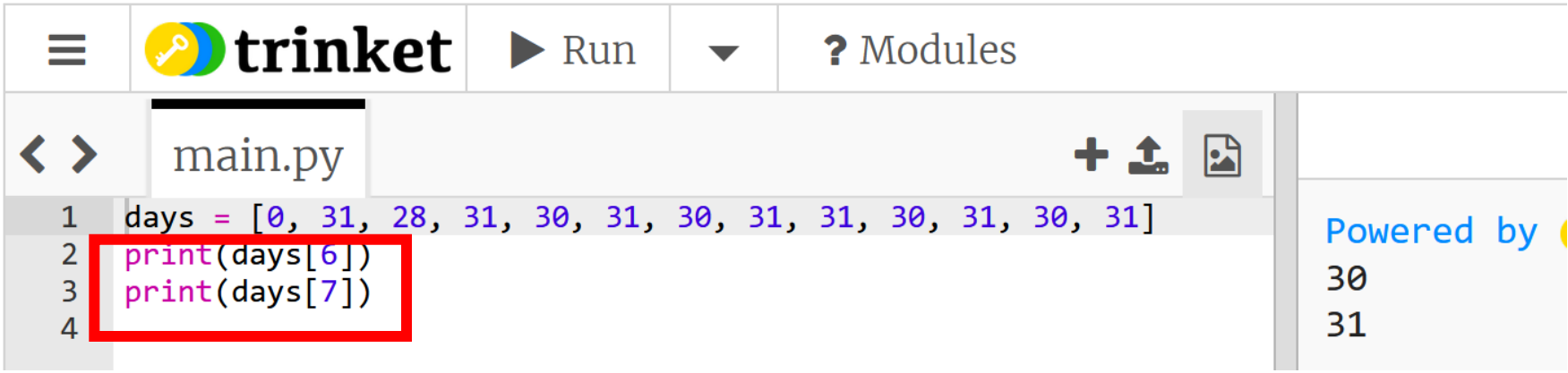
①trinketの次のページを開く

https://trinket.io/python/88a728c3cb

② 実行する．**30, 31が表示**されることを確認



③ ８月、９月を表示するように「print(days[6])」、「print(days[7])」を書き換えて実行する



print(days[6]) → 30（6月は30日）

print(days[7]) → 31（7月は31日）

0番目に空の要素を置くことで、月の番号とインデックスが一致する（days[6]＝6月）

演習（変更して実行）

print(days[8]) → 31（8月）
print(days[9]) → 30（9月）

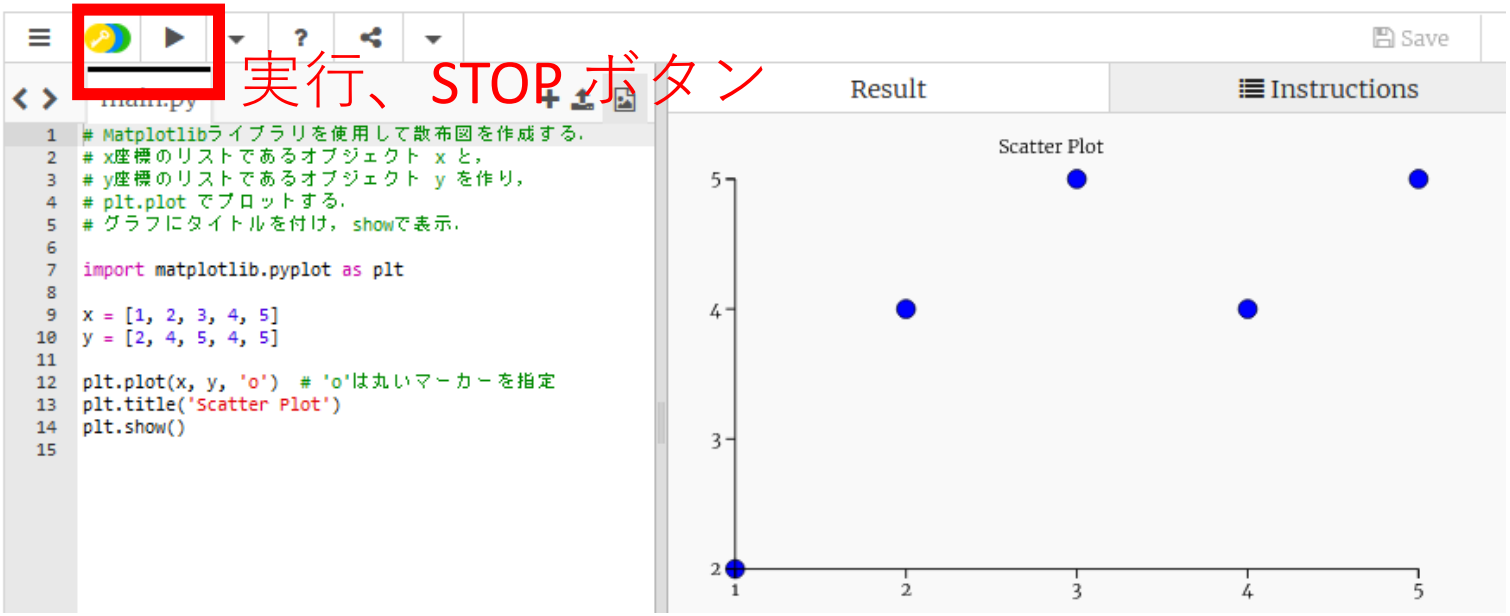
番号を変えれば、他の月の日数も表示できる

演習５．散布図。ライブラリの利用

① trinket の次のページを開く

https://trinket.io/python/a563124a187c

② 実行結果が，次のように表示されることを確認



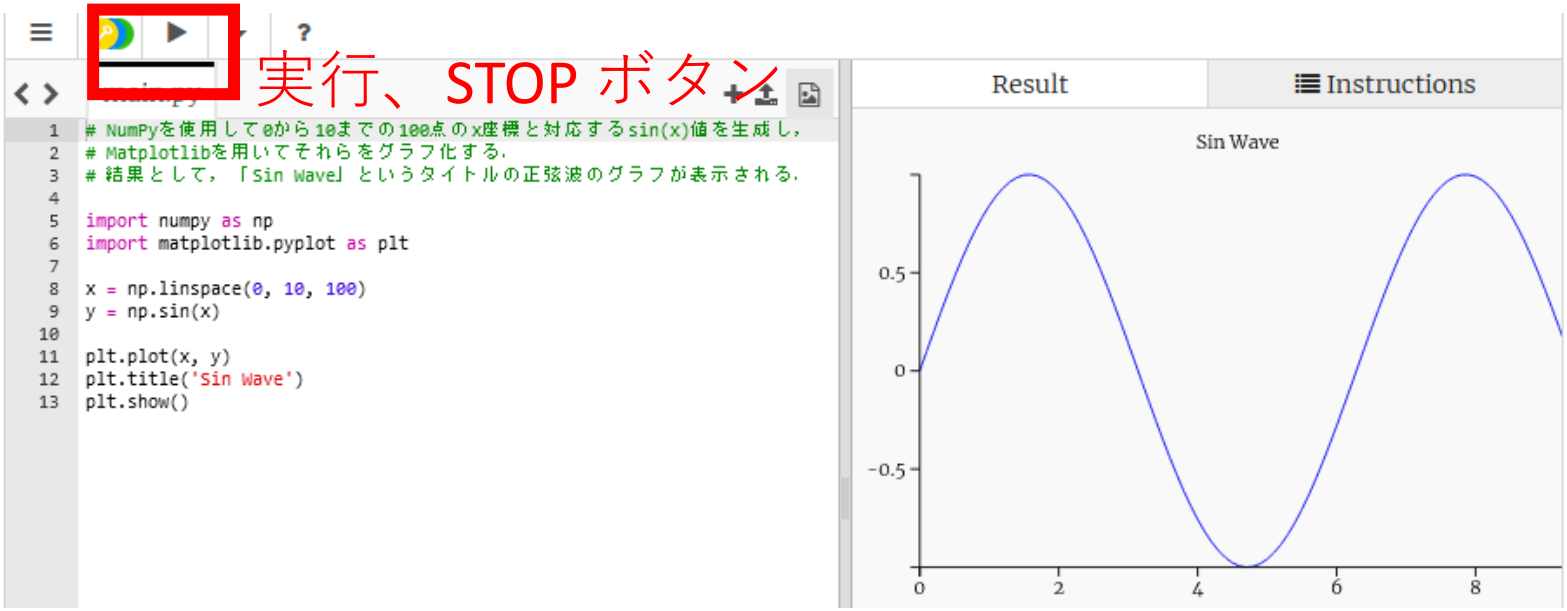
- 実行が開始しないときは、「**実行ボタン**」で**実行**
- プログラムを**書き替えて再度実行**することも可能

演習６．sin波．ライブラリの利用

③ trinket の次のページを開く

https://trinket.io/python/5830b6d18e9c

④ 実行結果が，次のように表示されることを確認



- 実行が開始しないときは、「**実行ボタン**」で**実行**
- プログラムを**書き替えて再度実行**することも可能

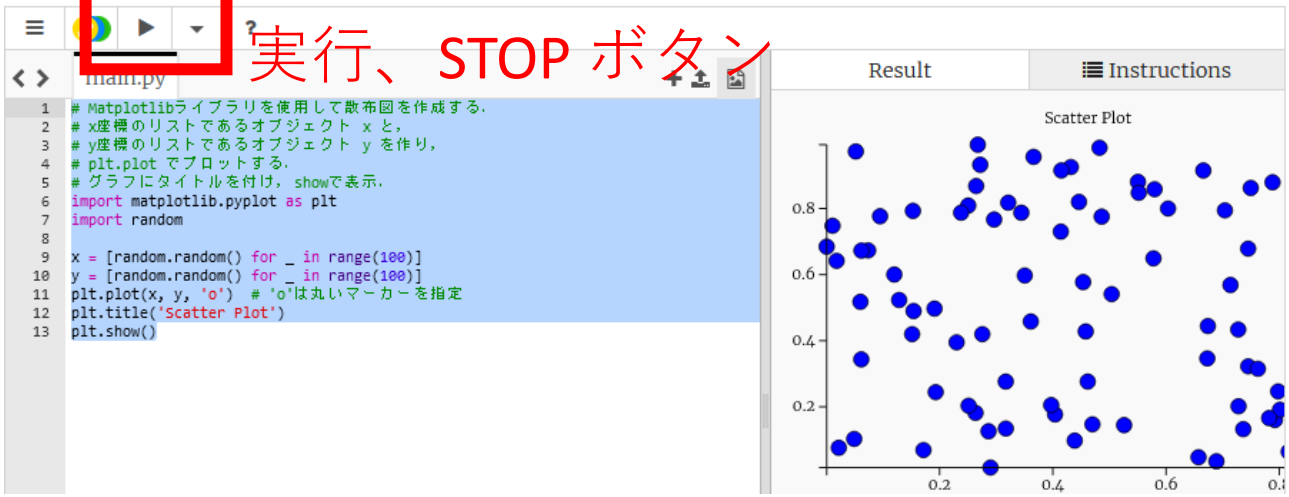
演習７．乱数．ライブラリの使用

① trinket の次のページを開く

https://trinket.io/python/9fe4ad1bb348

- 乱数（予測不可能な数値）の生成を確認．乱数は，シミュレーション、ゲーム、暗号化などで使用される重要な機能．

② 実行結果が，次のように表示されることを確認。**乱数で座標を生成している**




```
# Matplotlibライブラリを使用して散布図を作成する．
# x座標のリストであるオブジェクト x と，
# y座標のリストであるオブジェクト y を作り，
# plt.plot でプロットする．
# グラフにタイトルを付け，showで表示．
import matplotlib.pyplot as plt
import random

x = [random.random() for _ in range(100)]
y = [random.random() for _ in range(100)]
plt.plot(x, y, 'o')  # 'o'は丸いマーカーを指定
plt.title('Scatter Plot')
plt.show()
```

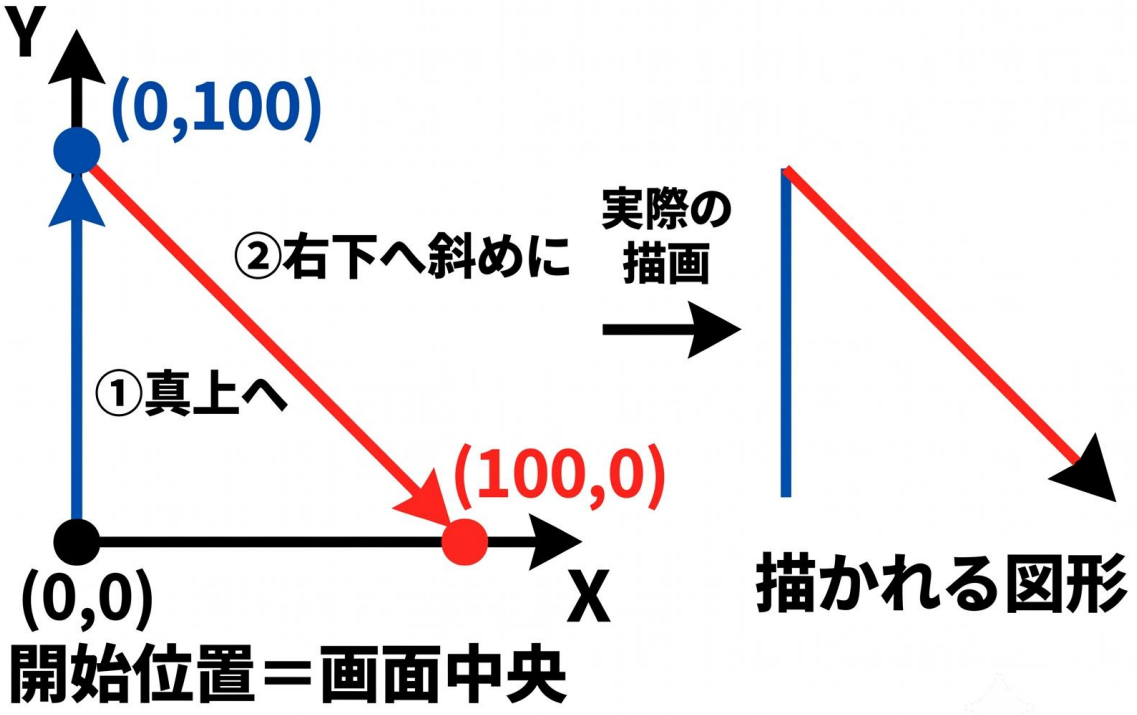
- 実行が開始しないときは、「**実行ボタン**」で**実行**
- プログラムを**書き替えて再度実行**することも可能

タートルグラフィックス

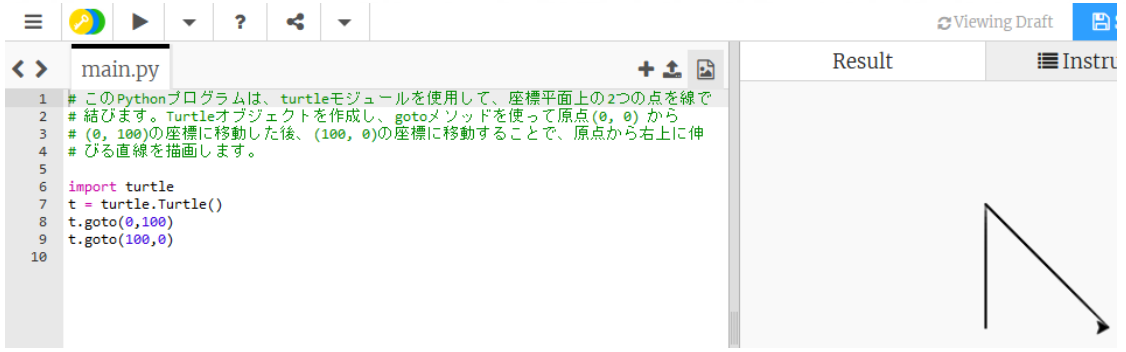
タートル（亀; turtle）が線を引きながら進む

```
import turtle
t = turtle.Turtle()
t.goto(0,100)
t.goto(100,0)
```

《Python プログラム》

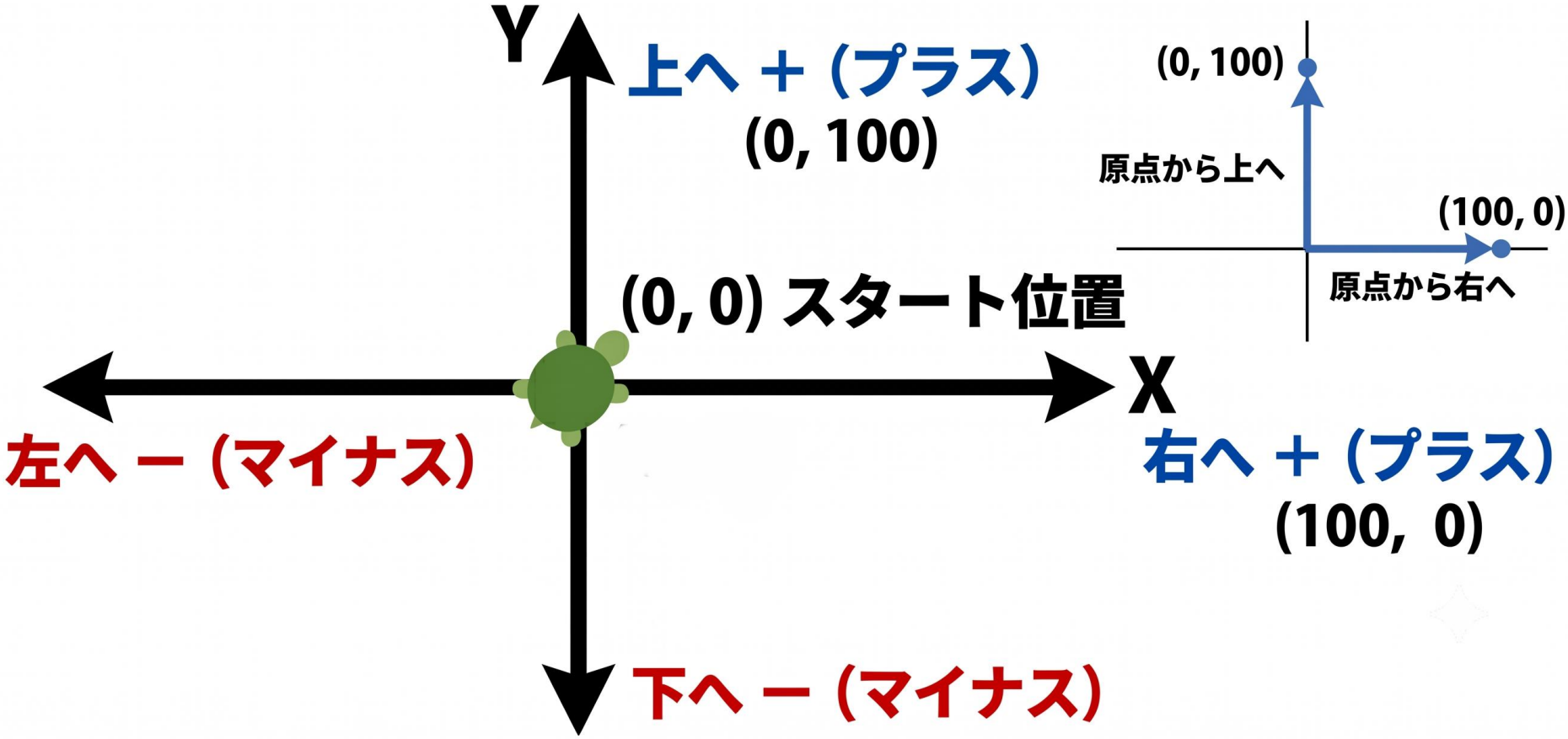


goto(x,y)：現在地から座標(x,y)まで線を引いて移動する



Goto メソッドでの場所の指定法

タートル（亀）の出発点は（0, 0）。そこを基準に上下左右へ位置が決まる



タートルグラフィックスのオブジェクトとメソッド

タートルの分身 t に命令（メソッド）を出して絵を描く

オブジェクトとメソッドの関係

オブジェクトメソッドの関係

```
t = turtle.Turtle()
t.goto(0,100)
t.forward(50)
```

t に対して点（.）でメソッドを呼ぶ

主要メソッドと移動対応

その他のメソッド

circle … 円を描く

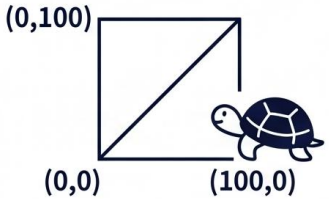
color … 線や塗りの色を変える

タートルグラフィックスで伸びる実力

turtleで図形を描く演習を通して、3つの学習姿勢を育てる

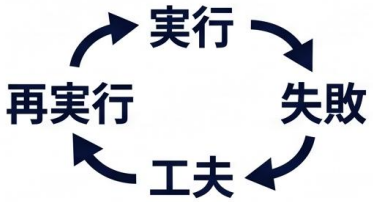
発想力・創造力

turtleで自分がデザインした図形を描く



自主性・自己研鑽

失敗は成長のチャンス。正解のない自由な課題に挑む



省察（ふりかえり）

工夫した点をふりかえると、さらに実力が伸びる

實力が育つ過程

自分でデザイン → 実行してみる → うまくいかない所を工夫 → ねばり強く改良 → デザイン＋実行＋粘り強い改良 ＝ 実力

演習８．タートルグラフィックス

①trinketの次のページを開く

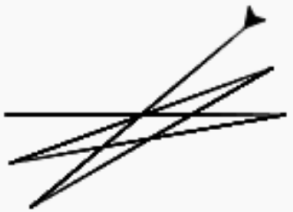
https://trinket.io/python/895c3ea5b6

② 実行結果が，次のように表示されることを確認

main.py

```
1  import turtle
2  t = turtle.Turtle()
3  for i in range(5):
4    t.forward(100)
5    t.right(170)
6
7
8
```

Result



③trinketの次のページを開く

https://trinket.io/python/0d8dbc1139

④ 実行結果が，次のように表示されることを確認

⑤ プログラム内の**「5」や「20」や「100」や「170」をいろいろ書き換えて実行**してみる

演習９．タートルグラフィックス

https://trinket.io/python/f8cd554693

```
import turtle

t=turtle.Turtle()

colors = ["red", "green", "blue"]

foriin range(3):

    t.color(colors[i])

    t.circle(30)

    t.forward(50)
```

ライブラリのインポート

オブジェクト生成．tへのセット．

色は，赤，緑，青

色を変える

半径３０の円

前に50進む

実行結果