

cs-8. Python プログラミング基礎①

(コンピューターサイエンス)

金子邦彦



「コンピューターサイエンス」第8回の内容



プログラミングはアイデアを形にする創造的作業。

変数 = 値 (オブジェクト) を指す名札



X = 20 + 30

左辺: 変数 (名札)

右辺: 式 (計算結果の値)

プログラムの基本要素

計算
演算子

出力
print()
画面に表示

入力
input()
値を受け取る

取り込み
import
外部機能を使う

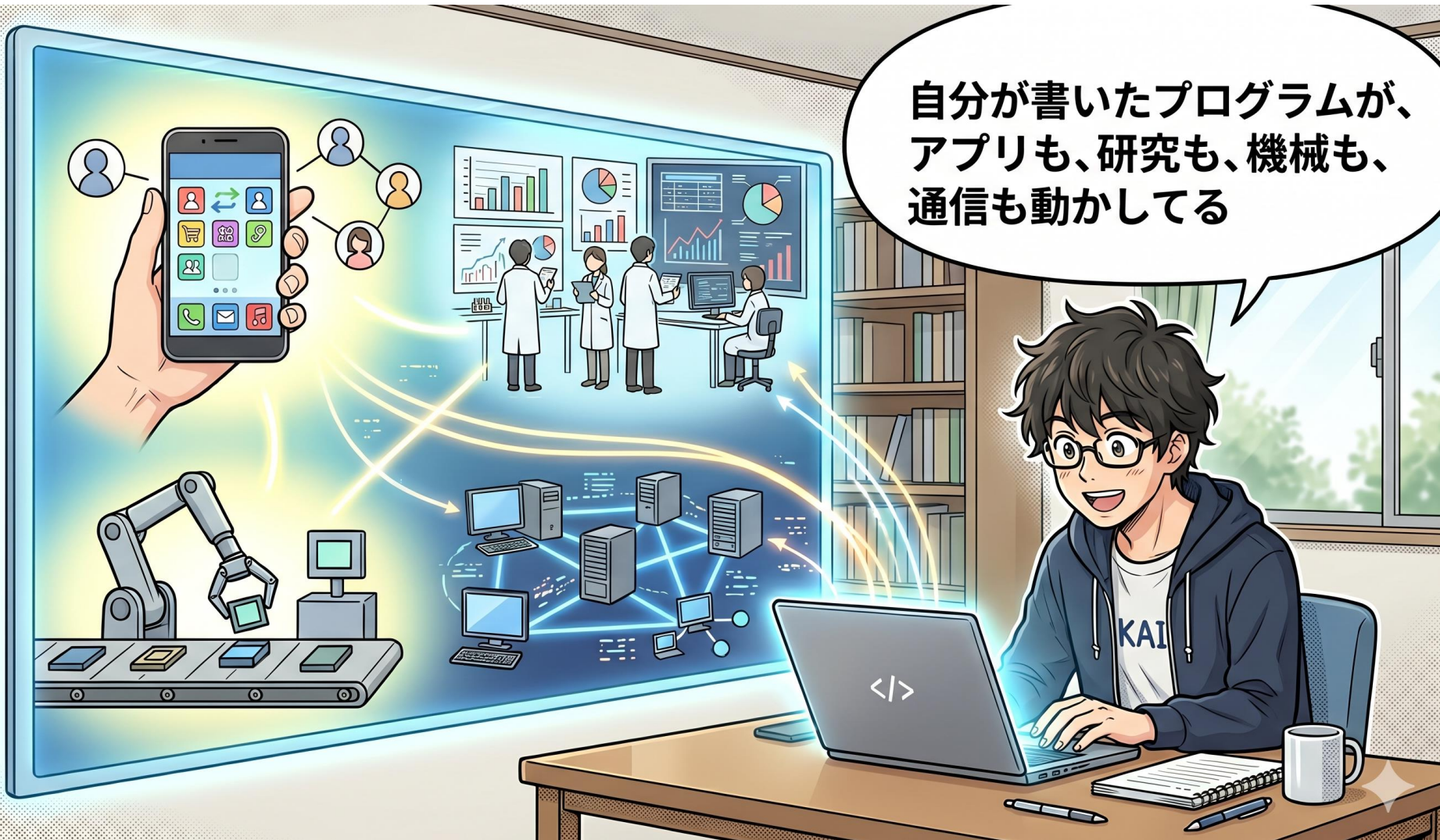


**タートル
グラフィックス**
図形描画を通して学ぶ

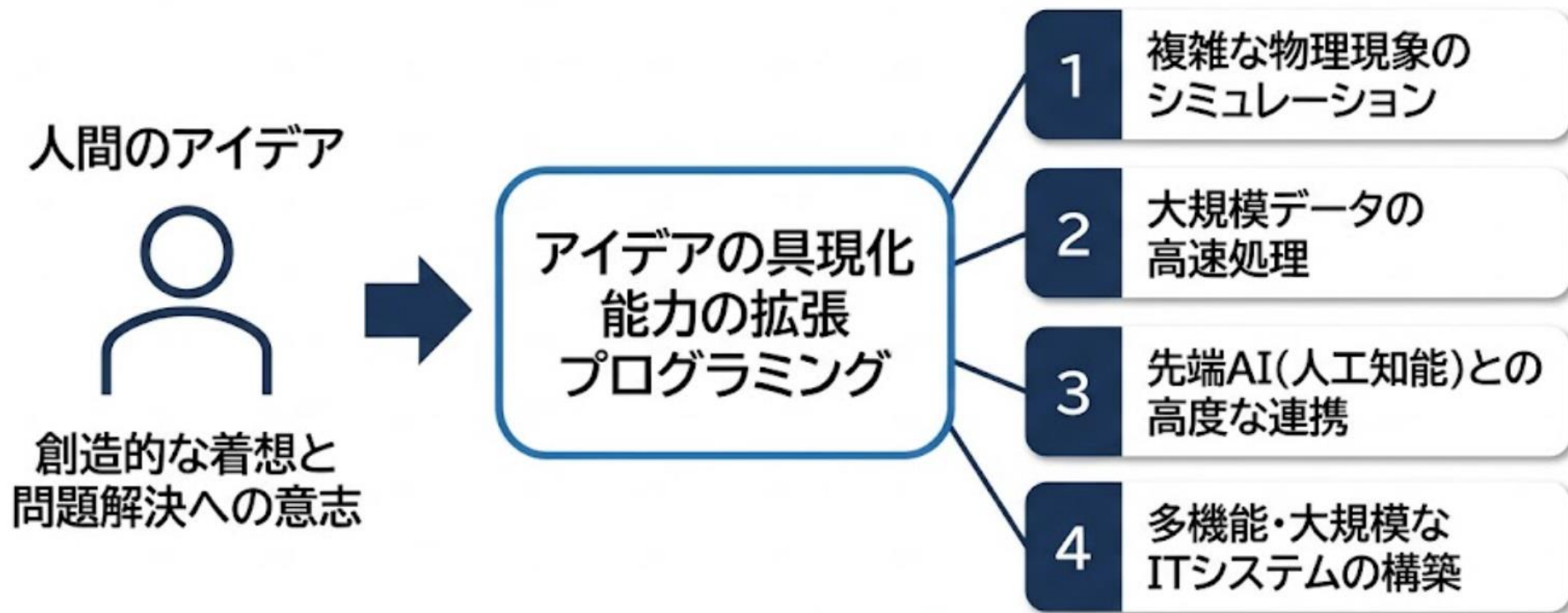
発想力
自主性
省察 (ふりかえり)

A square with four arrows pointing clockwise from the top, right, bottom, and left sides, representing a cycle or iteration.

8-1. プログラミングの全体像



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

プログラムとは — 命令を書いた手順の集まり



指示を与える

```
a = [200, 400, 300]
for i in a:
    print(i * 1.1)
```

自動で処理

200 / 400 / 300

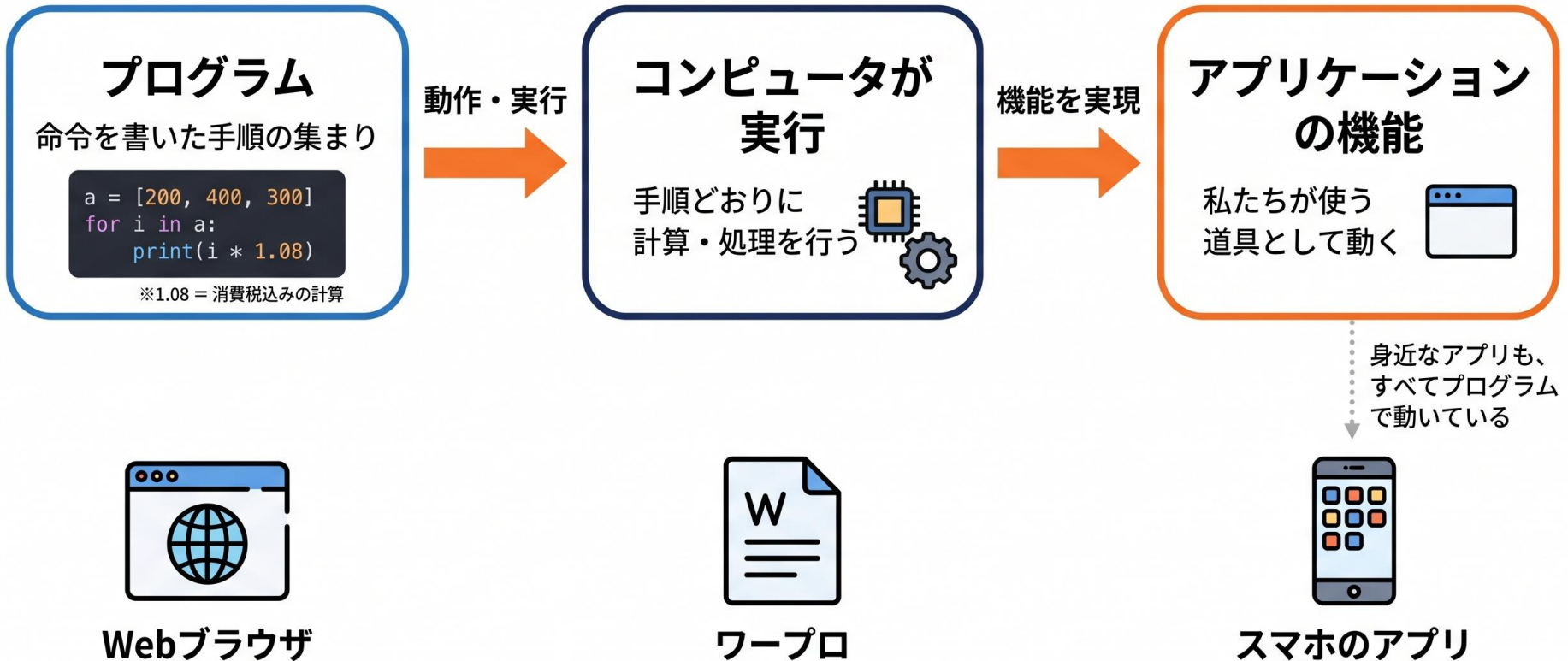


結果を得る

220.0 / 440.0 / 330.0

だから複雑な作業も **【自動化・効率化】** できる

プログラミングの用途① アプリケーションの実現



プログラムが動作することで、ブラウザやワープロなどアプリケーションの機能が実現される。

Python言語で記述する

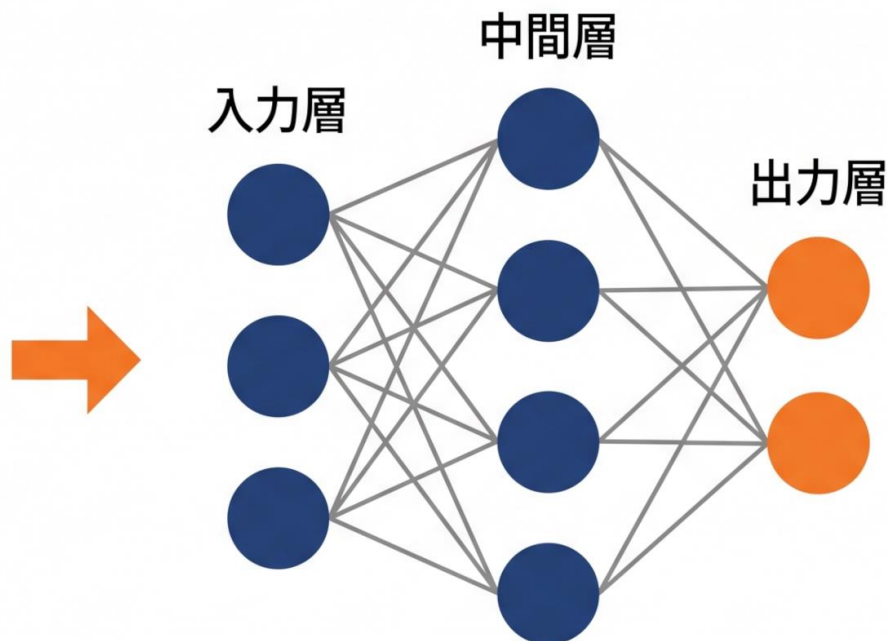
```
import torch.nn as nn

model = nn.Sequential(
    nn.Linear(3, 4),
    nn.ReLU(),
    nn.Linear(4, 2),
)

pred = model(x)
loss = loss_fn(pred, y)
loss.backward()
```

人間に読みやすい文法で
命令を記述

ニューラルネットワークで AIを構築する



データから規則を学習し、
判断・予測を行うAIシステム

プログラムが コンピュータを制御する

プログラム（命令を書いた手順の集まり）



命令を実行

コンピュータ

CPU

制御・演算

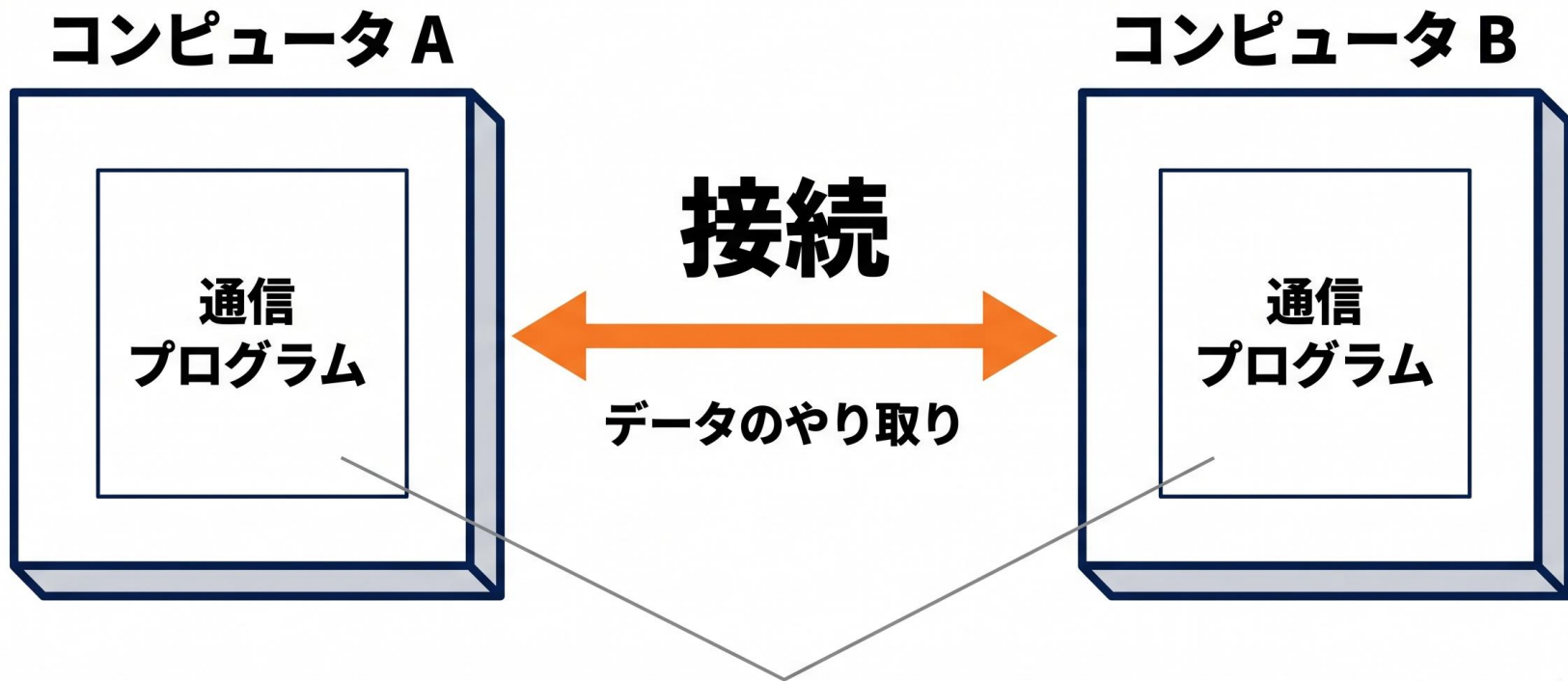
メモリ

記憶

入出力

入力・出力

コンピュータ同士の接続にはプログラムが必要



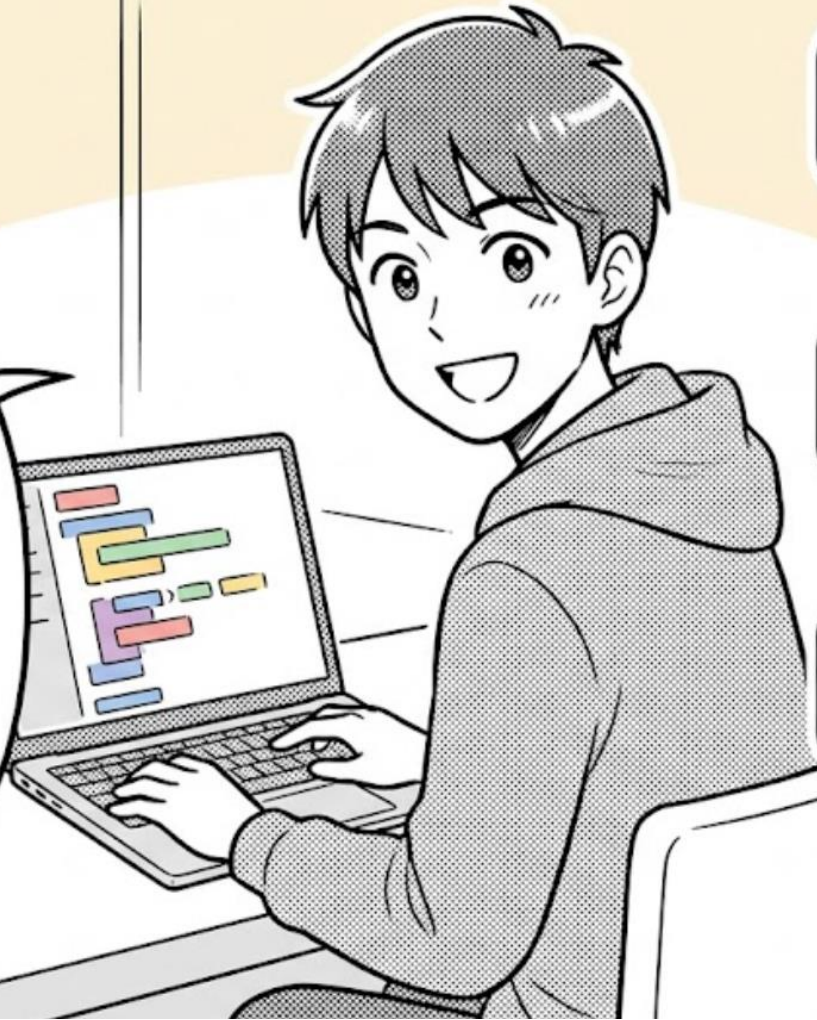
接続を行う処理そのものをプログラムが担当する
相手とやり取りする手順（ルール=プロトコル）に従ってプログラムが動作する



8-2. Python 言語とツール



ソースコードは
人とコンピュータの
“共通のことば”。
読めて、書けて、
あとから直せる。
だから扱いやすい!



書く



実行



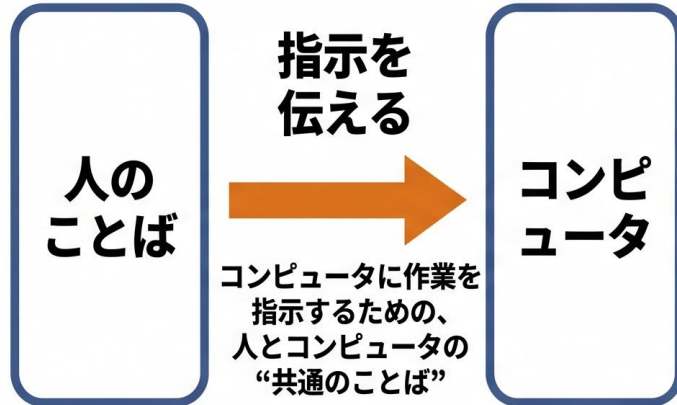
結果



Python (パイソン) 言語



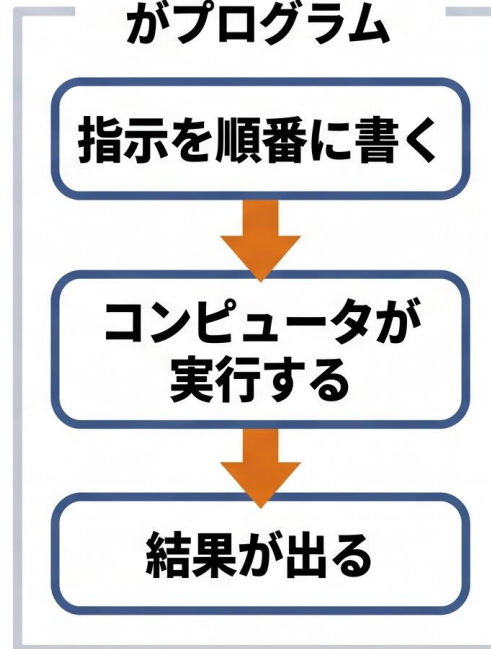
プログラミング言語とは



その言語の一つが Python

プログラムとは

指示を並べた“手順書”
がプログラム



Python の特徴

読みやすい

— 英語の文章に近く、
見て意味が
分かりやすい

書きやすい

— 少ない記述で
動かせる

応用が広い

— 簡単な処理から
高度な開発まで対応

シンプルな用途から高度なプログラムまで、はば広く使える

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

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

Python 例

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

編集・修正

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

人間が読める言葉
(プログラミング言語)で
書かれている

実行での流れ



コマンドプロンプトによるPython実行



WindowsでコマンドプロンプトからPythonを実行する場合

- Pythonのインストール (<https://www.python.org> から)
- **python**コマンドでPythonを起動すると、**プログラムを入力するたびに結果が得られる対話的実行が可能**
- 終了は**exit()**コマンド

```
C:\Users\user>python
Python 3.10.9 (tags/v3.10.9:1dd9be6, Dec 6 2022)
Type "help", "copyright", "credits" or "license()"
>>> x = 100
>>> if (x > 20):
...     print("big")
... else:
...     print("small")
...
big
>>> s = 0
>>> for i in [1, 2, 3, 4, 5]:
...     s = s + i
...
>>> print(s)
15
>>>
```

Python
プログラム
実行
結果

コマンドプロンプトで Pythonで開始

- Windowsでは、**python**コマンドで実行
- 終了は **exit()**

Python プログラムのパソコン上での実行



Python プログラムはオンライン実行（例：Trinket）のほか、パソコンでも実行可能。（パソコンでの実行の場合には、Python 処理系のインストールが必要）

① Python プログラムのファイル保存

```
x = 100
if (x > 20):
    print("big")
else:
    print("small")
s = 0
for i in [1, 2, 3, 4, 5]:
    s = s + i
print(s)
```

作成した **Python プログラム** の **ソースコード** を、例えば「foo.py」という名前の **ファイル** に保存

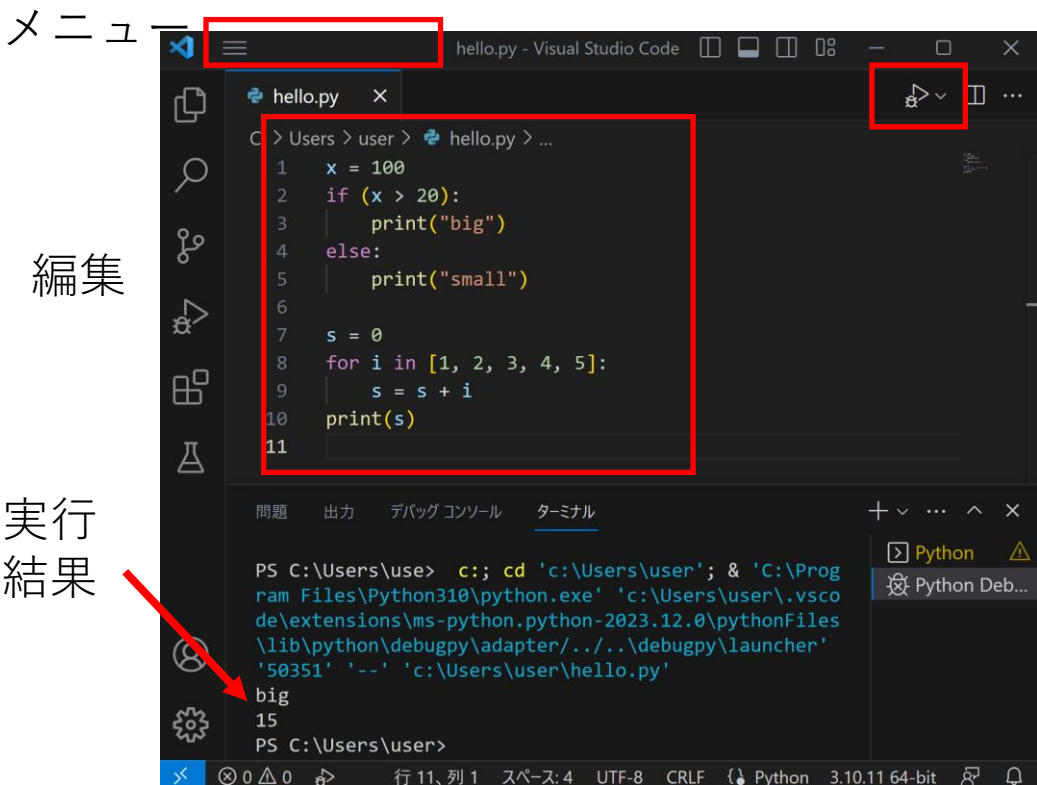
② Python プログラムの実行

```
kaneko@www:/tmp$ python foo.py
big
15
```

プログラムを実行するには、シェル（例えば、Windows の場合はコマンドプロンプト）を開き、「python foo.py」のようなコマンドで実行

Visual Studio Codeによる開発

- Visual Studio Codeは、Microsoftが開発した多言語対応の統合開発環境
- Pythonのインストール (<https://www.python.org> から)
- Visual Studio Codeのインストール (<https://www.microsoft.com/ja-jp/dev/products/code-vs.aspx>) が必要
- 編集画面でプログラムを編集し、ターミナル（端末）で実行結果を確認できる。デバッグ機能により変数探索も可能である。



実行ボタン

※ 「デバッグ」の機能により変数探索も可能

オンライン実行環境の使いわけ

目的：手軽に即実行

OneCompiler

ブラウザだけで即座にプログラムを実行し、結果を確認できる

ビジュアルプログラミングにも対応

GDBonline

ブラウザだけで即実行・結果確認ができる

オブジェクト観察やステップ実行など、学習用機能が充実

目的：本格的なAI・データ処理

Google Colab

無料で利用でき、ファイル共有も可能。
AIアシスタント付き
Googleアカウントがあれば、AIや3次元処理など本格的なプログラムも実行可能

OneCompiler / GDBonline で手軽に

この2つは、この授業で使用



Google Colab で本格的に

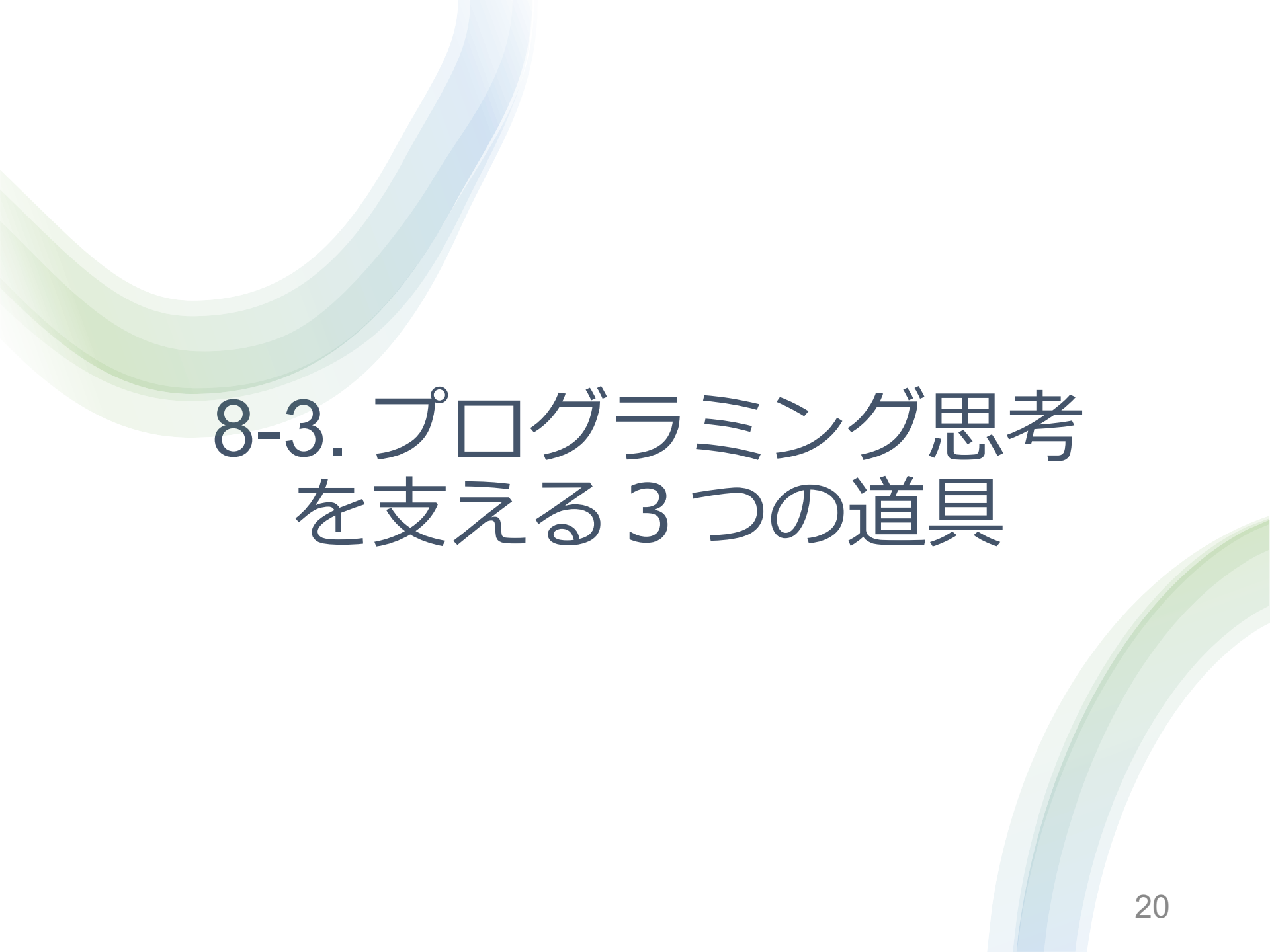
別の授業で紹介。自力で調べることも十分に可能

Google Colaboratory

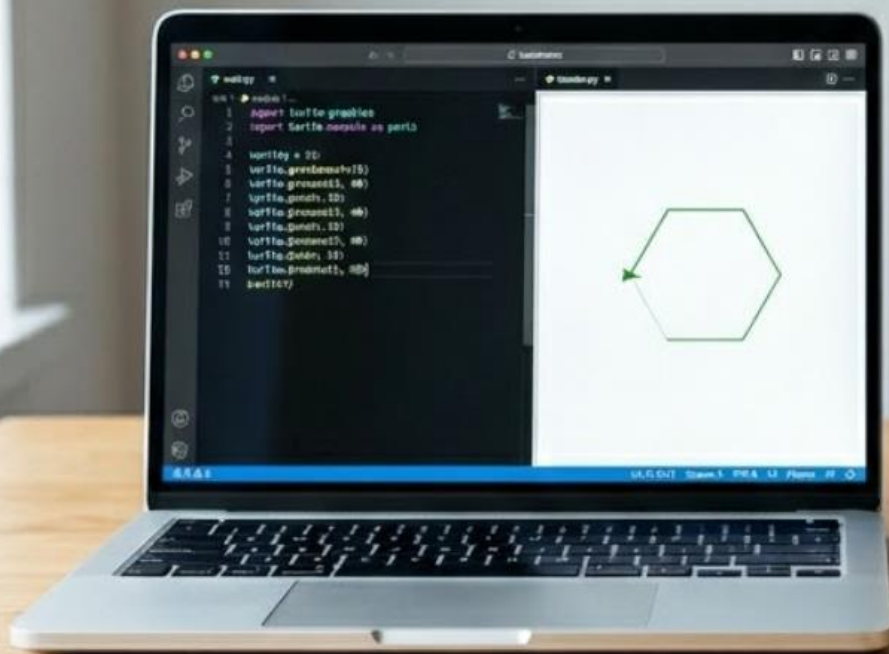


Google Colaboratory <https://colab.research.google.com/>

- **Python の開発環境**
- 人工知能，データサイエンス，その他多数のパッケージがインストール済み
- **コードセル，テキストセルを複数ノートブックにまとめ，保存や公開できる**
- **ノートブックにより，記録が簡単に残せる．ビジュアルな表示も簡単に可能**
- **プログラムの共有も簡単**



8-3. プログラミング思考 を支える3つの道具



プログラミング思考を支える3つの道具：①オブジェクト・メソッド・引数、
②変数と代入、③アルゴリズムの感覚

プログラミング思考を支える3つの道具

複雑な問題を解くための土台となる基本概念

①

**オブジェクト・
メソッド・引数**

操作する対象と、
その動作、そして動
作に渡す追加情報

②

変数と代入

データに名前を付
けて保管し、必要に
応じて書き換える

③

アルゴリズムの感覚

問題を解く手順を
順序立てて組み立て
る直感

処理を表す三点セット

turtle . **goto** (0, 100)

オブジェクト

処理の対象を
表すもの

メソッド

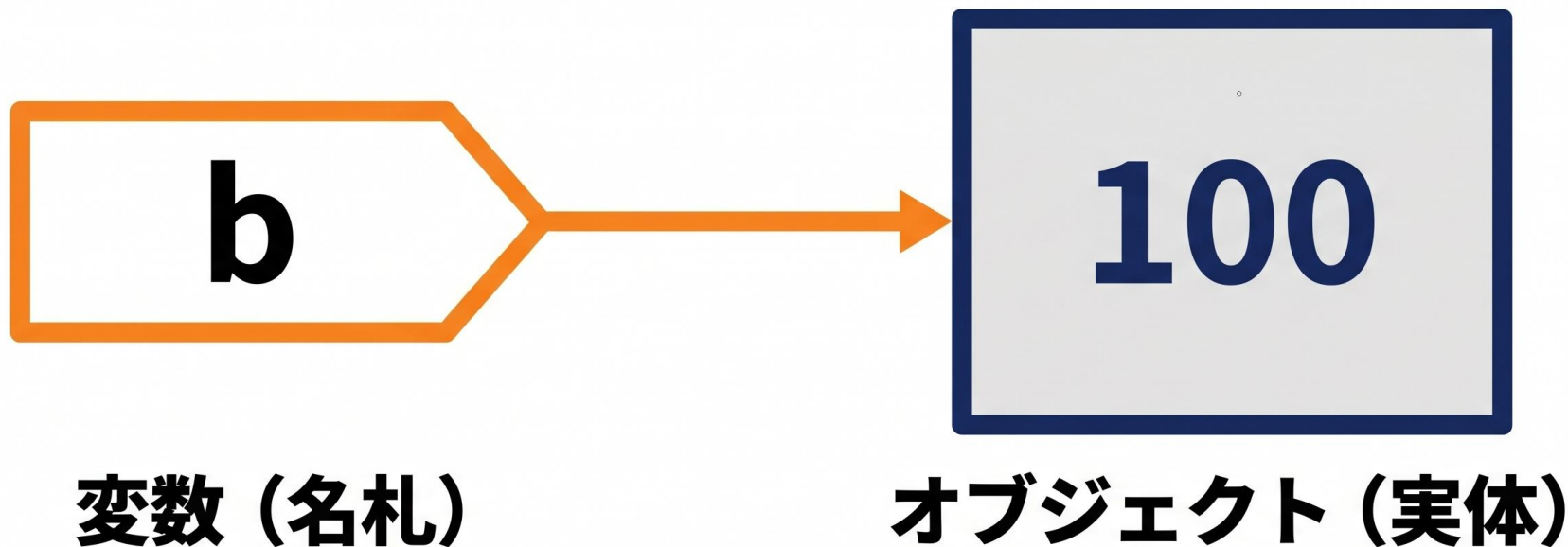
オブジェクトの
能力を表すもの

引数

能力を細かく
指定するもの



変数は『名札』、オブジェクトは『実体』。名札が実体を指し示す。



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

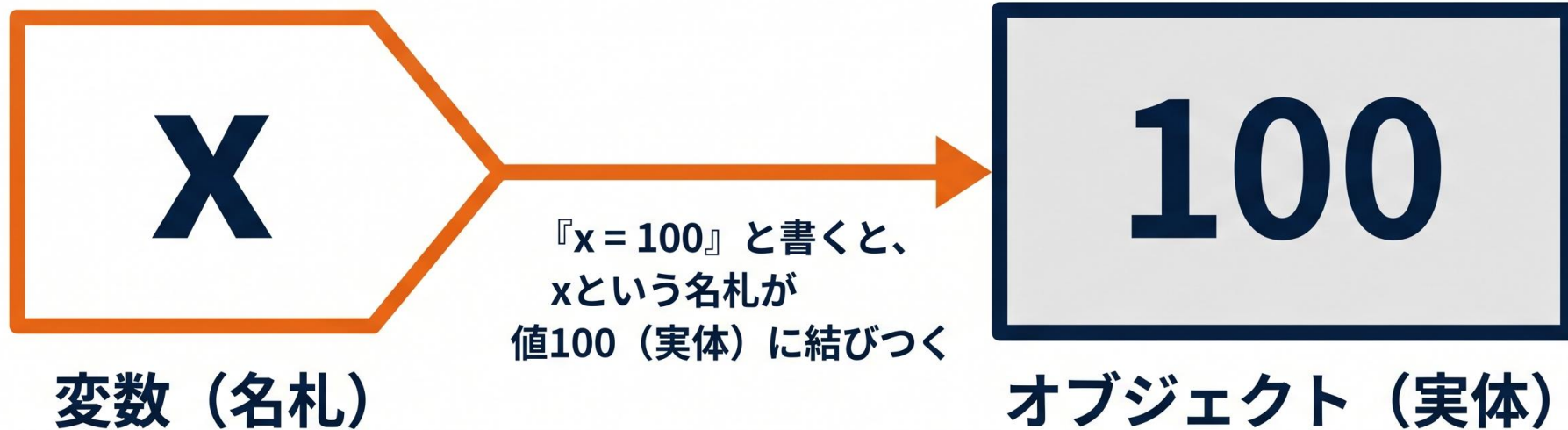


変数と代入



代入とは、名札（変数）に実体（値）を結びつけること

$$x = 100$$



再代入



再代入：今の値をもとに計算した結果を、同じ変数にもう一度入れ直す



$$x + 100$$

(入れ直す)



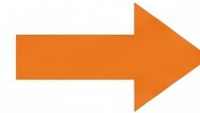
今のxの値に100を足す

入れ直した値



$$\text{score} + 1$$

(入れ直す)



今のscoreに1を足す

入れ直した値

代入文の構文ルール



代入文は『**左辺=変数名、右辺=式**』の順で書く

正しい (○)

X = X * 2

左辺：変数名 代入 右辺：式

誤り (×)

2 * X = X

左辺が式になっている → 不可

左辺は必ず変数名でなければならない。右辺の計算結果を左辺の変数に入れる。

アルゴリズムの感覚：手順を順序立てて組み立てる



```
import turtle  
t = turtle.Turtle()
```

```
t.forward(50)  
t.left(60)  
t.forward(50)  
t.left(60)  
t.forward(50)  
t.left(60)  
t.forward(50)  
t.left(60)  
t.forward(50)  
t.left(60)  
t.forward(50)  
t.left(60)  
t.forward(50)  
t.left(60)
```

① やりたいこと（ゴール）



正六角形を描きたい

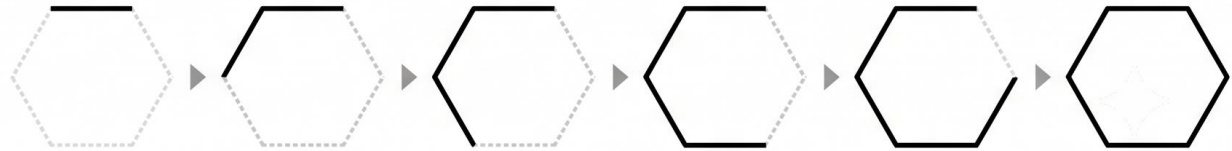
② 小さな手順に分解

前進する（50すすむ）

左に曲がる（60度）

この2つを6回くりかえす

③ 順番に実行すると形になる



六角形の辺が1本ずつ増えて完成する

8-4. 式・計算・演算子、 データの種類

式を実行すると、計算されて1つの値が得られる

式例→結果

$$3 + 2 \rightarrow 5$$
$$10 - 4 \rightarrow 6$$

実行する

$$6 * 4 \rightarrow 24$$
$$20 / 5 \rightarrow 4.0$$

掛け算

$$5 * 2 \rightarrow 10$$

2倍は * 2

$$7 * 2 \rightarrow 14$$

割り算

/

$$8 / 2 \rightarrow 4.0$$

半分は / 2

$$10 / 2 \rightarrow 5.0$$

式は計算して結果を返す。書くだけでは保存されない

いろいろな式

$$3 + 4 \rightarrow 7$$

$$10 - 2 \rightarrow 8$$

$$6 * 5 \rightarrow 30$$

$$15 / 4 \rightarrow 3.75$$

$$15 // 4 \rightarrow 3 \text{ (整数の割り算)}$$

$$15 \% 4 \rightarrow 3 \text{ (余り)}$$

$$2 ** 3 \rightarrow 8 \text{ (べき乗)}$$

式は結果を返すだけ

```
print(3 + 4)
```

計算した結果は表示されるが、
どこにも保存されない

↓
7

保存するには代入する

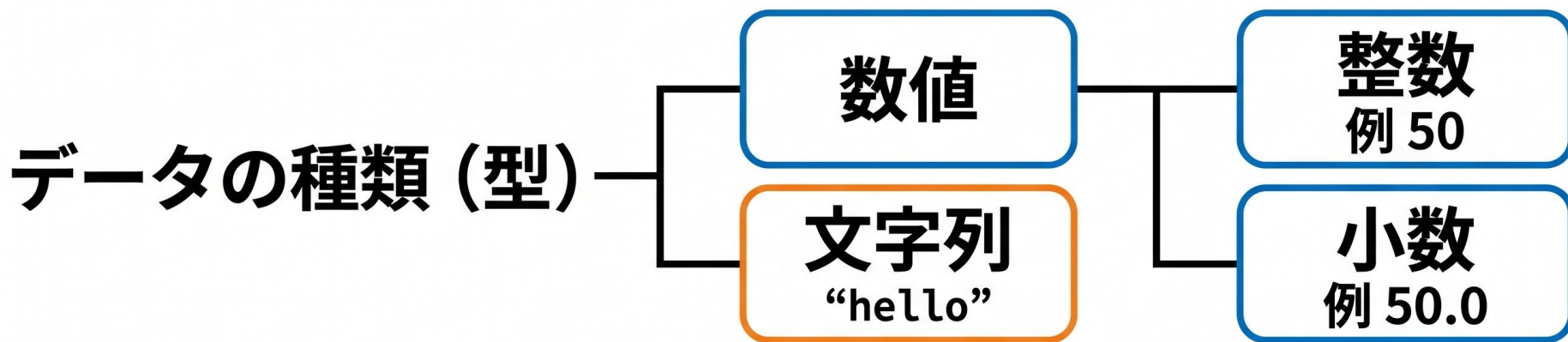
```
3 + 4
```

計算するだけ。消えてしまう

```
a = 3 + 4
```

結果が **a** に保存される
(a は 7)

データには種類 (型) がある



同じ $x = 100$ でも、計算によって結果の型が変わる

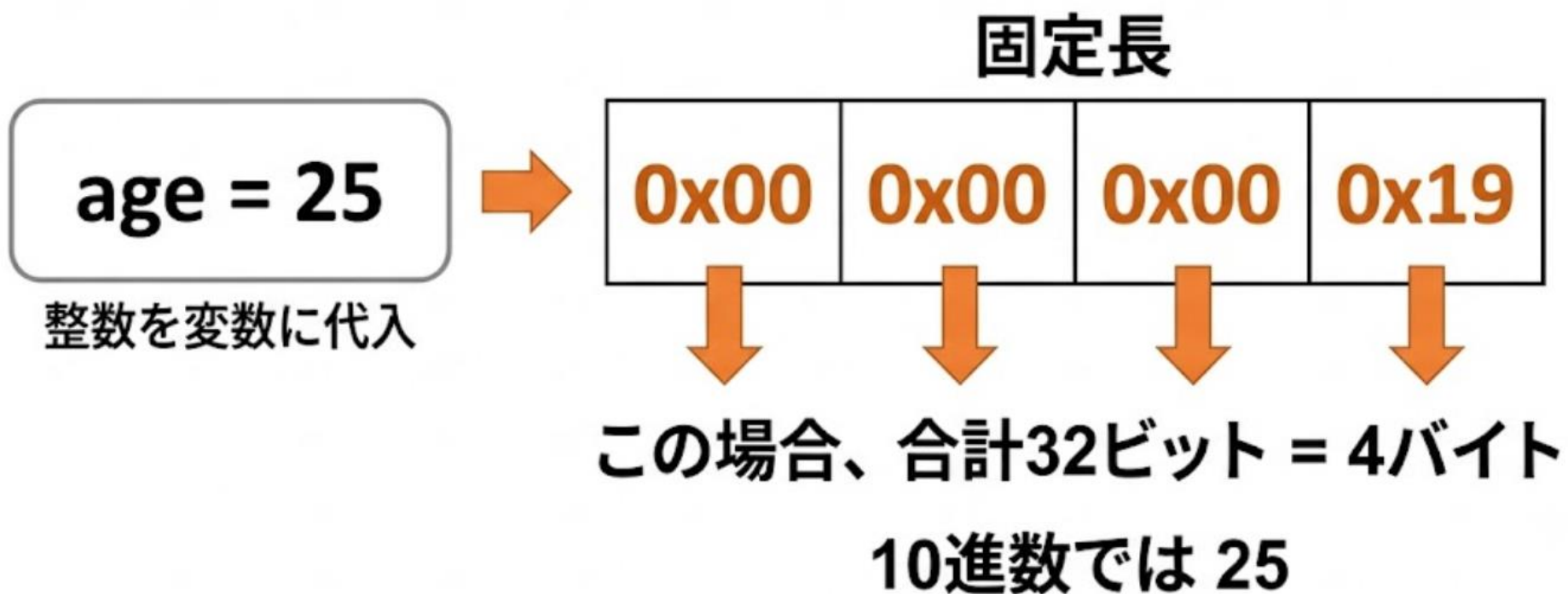
$x / 2$
↓
50.0
小数

$x * 2$
↓
200
整数

`math.sqrt(7)`
↓
2.645...
小数

割り算 (/) は結果が割り切れても小数になる

整数は内部では決まった大きさ（固定長）で保存される

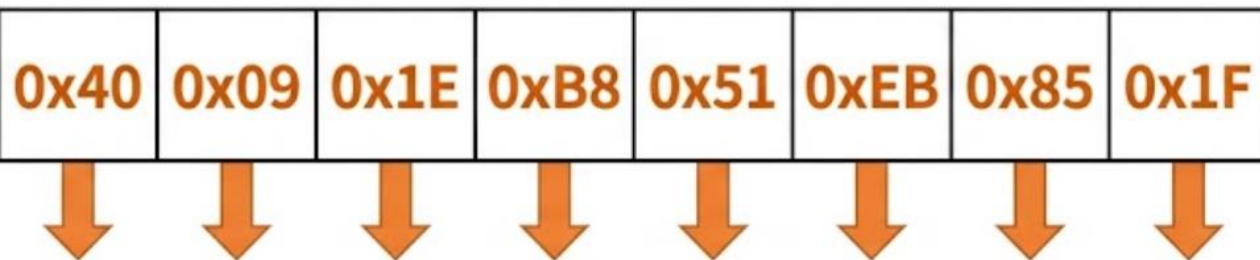


小数は内部では決まった大きさ（固定長）で保存される

固定長

price = 3.14

小数を変数に代入



この場合、合計64ビット = 8バイト

10進数では 3.14

文字コード（文字の数値表現）



文字は文字コードによって数値（2進数）に変換されて扱われる

文字	2進数（8ビット）	16進数
A	01000001	41
B	01000010	42

文字コード (ASCII) で数値化

ASCIIは7ビット → 先頭に0を補い8ビットにする

7ビット表現：1000001 → 8ビット：

0	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---

補う0

コンピュータはデータを8ビット(1バイト)単位で扱うため、7ビットのASCIIに先頭0を補う

Python での文字列データ

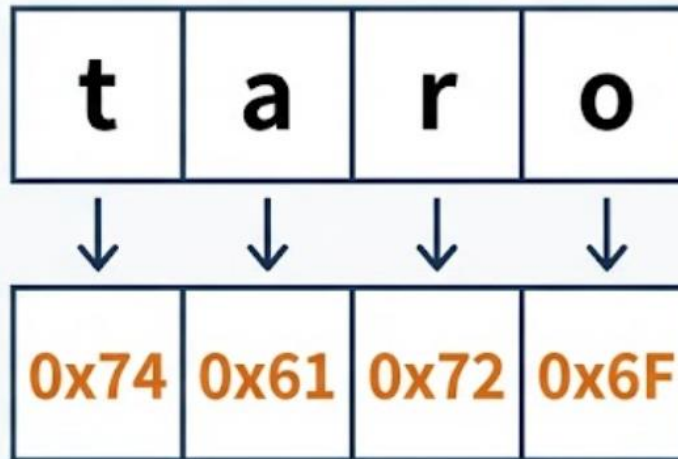


文字は内部では数値（文字コード）として保存されるから、表示も連結もできる

```
name = "taro"
```

文字列を変数に代入

内部では数値（文字コード）



10進=116, 97, 114, 111

```
print("taro")
```

→ 画面に「taro」と表示

```
「私は」 + name
```

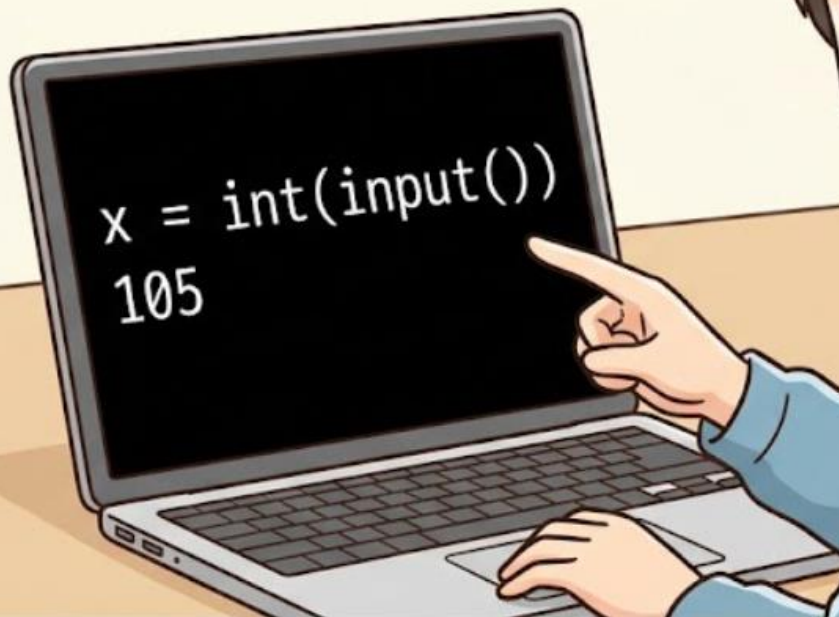
→ 文字列の連結ができる

1文字ずつ番号（文字コード）が決まっている
ここでは16進数で表記

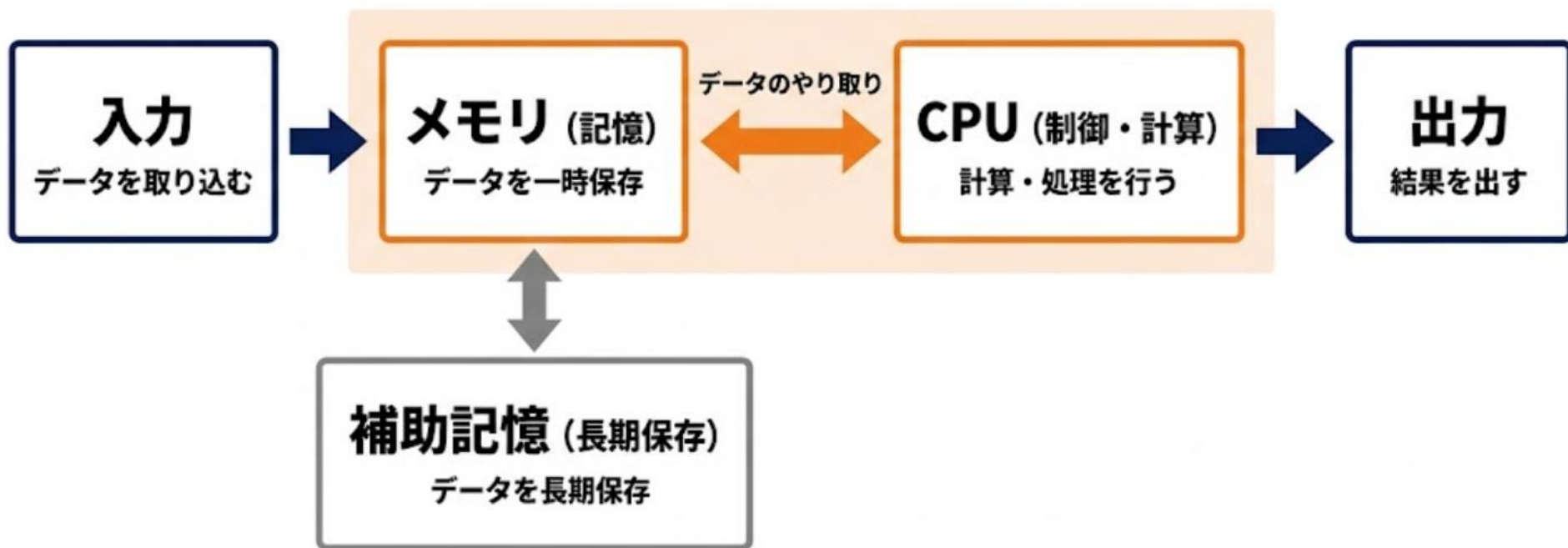
数値として一貫して扱えるから成立する

8-4. コンピュータの中で 起きていること

キーボードの『5』は文字だけど、
int()で整数に変わって、
+100されて『105』になるのか！



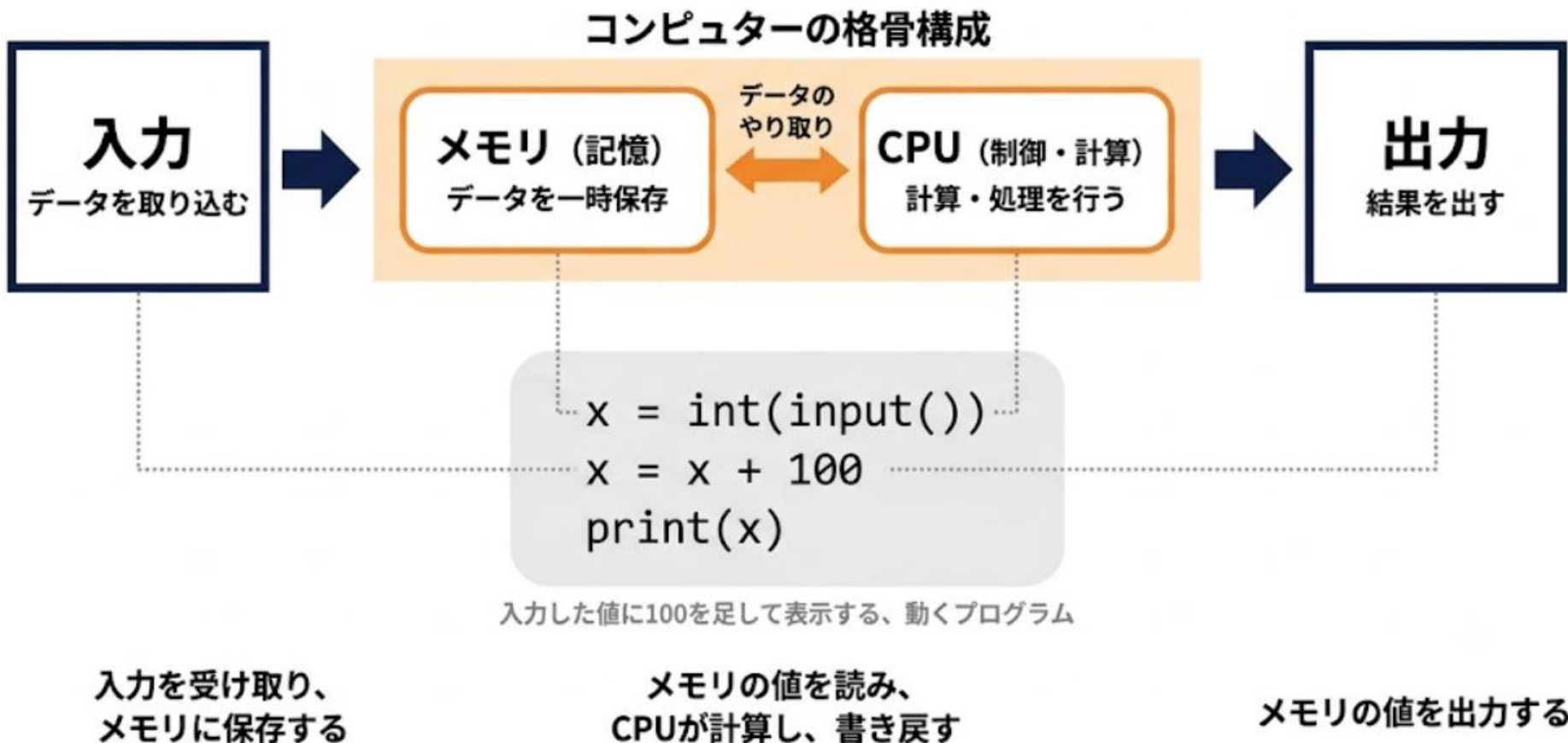
コンピュータは入力・メモリ・CPU・出力が
つながり、データを順に受け渡して動く



入力・計算・出力



Pythonの input・計算・print は、メモリとCPUのやり取りそのもの

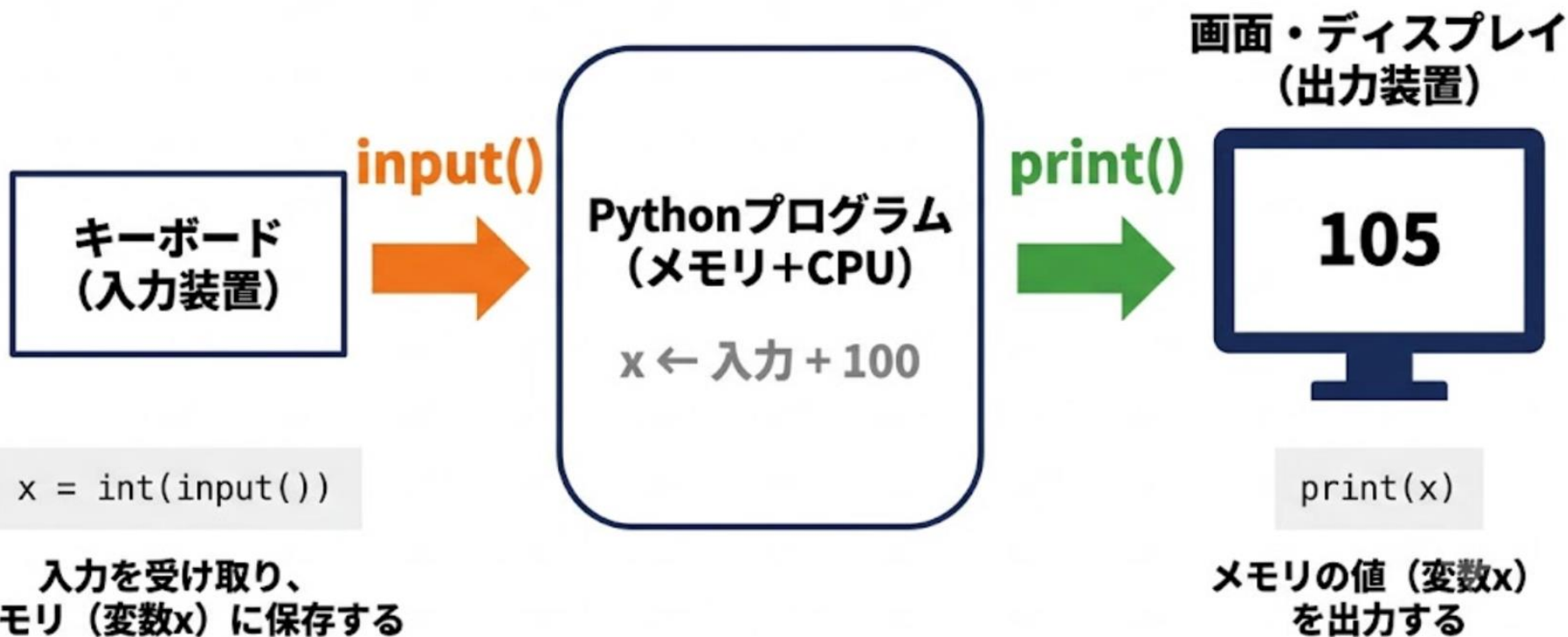




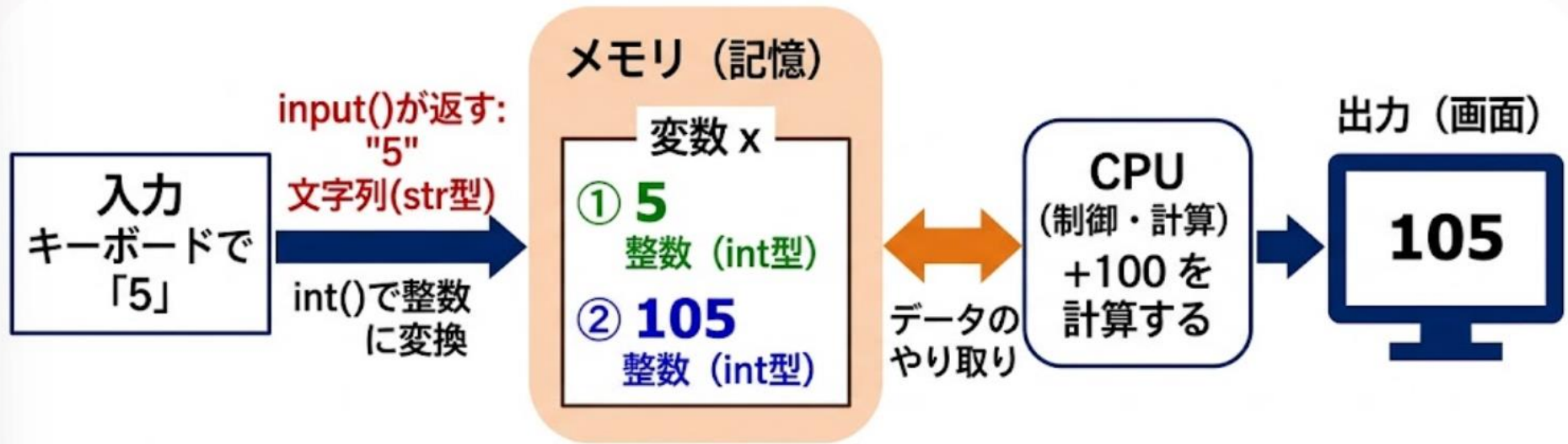
入力・計算・出力

input() で入力。メモリの保存。
print() で出力

```
x = int(input())  
x = x + 100  
print(x)
```



入力・計算・出力 — 変数とデータの種類



```
x = int(input())
```

キーボードの「5」を input() が文字列"5"として受け取り、int()で整数5に変換してから箱xへ保存 (→①)

```
x = x + 100
```

CPUが箱xの5を読み、100を足して105を書き戻す (①→②)

```
print(x)
```

箱xの105を画面へ出力する

input() による入力



input() は必ず文字列で受け取る — 計算したいときは int() で整数にする

① そのまま受け取る

キーボードで「5」と打つ

```
x = input()
```

input()が文字列を
返し、そのままxへ

変数 x
"5" (文字列 / str型)

input()の結果は必ず文字列。このxは計算できない

② 整数に変換して受け取る

キーボードで「5」と打つ

```
x = int(input())
```

input()が返す"5" (文字列)
int()が整数に変換

変数 x
5 (整数 / int型)

変換後の整数だけがxに入る。だから計算できる

同じ「5」でも、①は文字列、②は整数。xに入るのは変換の結果

input() による入力



なぜ int() が必要？ — 文字の「5」は、そのままでは計算できない

```
x = int(input())
```

キーボードで「5」を押す

文字コードとして送られる

'5' = 文字コード 53 (0x35)

input() が受け取って返す

"5" 文字列 (str型)

input() の戻り値。まだ変数 x には入っていない。このままでは計算できない

int() が整数に変換する

5 整数 (int型)

変換が終わって初めて変数 x に代入される

ここで初めて x に値が入る

ここで初めて x に値が入る

変数 x = 5 → x + 100 = 105

print() は、文字列や変数の値を画面に表示する命令

① 文字列を表示する

```
print("こんにちは")
```



こんにちは

画面 (出力)

② 変数の値を表示する

x = 105

```
print(x)
```



105

print() による出力：文字列と値を並べる



文字列と値を並べる：カンマ「,」とプラス「+」の使い分け

$x = 105$

カンマ「,」で区切る

```
print("x =", x)
```



x = 105

カンマで区切ると、間に半角スペースが自動で入る

プラス「+」で連結する

```
print("x = " + str(x))
```



x = 105

文字列どうしを直接つなぐ
(数値は str() で文字列にする必要がある)

2つの出力結果はどちらも『x = 105』になる

8-7. ライブラリとインポート

import って何? —— 使える機能を増やす 1 行

標準機能

print()
if 文
for 文
リスト

Python をインストールした時点で、
import なしですぐ使える機能

import numpy

import matplotlib

この 1 行で、新しい機能が
使えるようになる

import turtle

ライブラリ = 特定の機能をまとめた
もの。import で読み込んで使う

NumPy

数値計算・配列を扱う

外部ライブラリ (自分で追加してから import で読み込む)

Matplotlib

データをグラフにする

外部ライブラリ (自分で追加してから import で読み込む)

turtle

図形を描いて動かす

標準ライブラリ (最初から入っているが、import で読み込む)

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



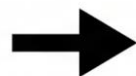
例：math ライブラリの sqrt (平方根) を使う

① ライブラリを
インポート (import)
で読み込む

```
import math
```

math ライブラリを
読み込む

読み込む
と使える



② 機能を
『名前.機能()』で呼ぶ

```
math.sqrt(9)
```

① で読み 使いたい 機能に
込んだ名前 機能(平方根) わたす値

実行
すると



③ 結果が返る

3.0

9 の平方根は 3.0

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

datetime ライブラリ



パソコンの時計を読み取って、今の日時を表示する
(datetime ライブラリ使用)



読み取る



```
import datetime
datetime.datetime.now()
```

結果を返す



2026-06-04 14:30:05

パソコン内部の時計
常に時を刻んでいる

ライブラリを呼び出す

今の日時が表示される

import datetime → datetime ライブラリの取り込み .now() → 実行した瞬間の時刻

mathライブラリ：Pythonで数学の計算を行うための標準ツール

使う前の準備

```
import math
```

まず最初に1回だけ書く。
これでmathの機能が
使えるようになる

基本の3つの機能

平方根
`math.sqrt(9)`
→ 3.0
ルートの計算

円周率
`math.pi`
→ 3.14159...
πの値 (定数)

三角関数
`math.sin(x)`
→ サインの値
xはラジアンで指定

角度の変換

角度 * `math.pi / 180`

度 (degree)

ラジアン (radian)

30度 → `30 * math.pi / 180` → 約0.523 (ラジアン)
`math.sin(30 * math.pi / 180)` → 0.5

三角関数に渡す前に、**度をラジアンへ変換**することを忘れない

math ライブラリの三角関数は角度をラジアンで受け取るため

8-8. タートルグラフィックス

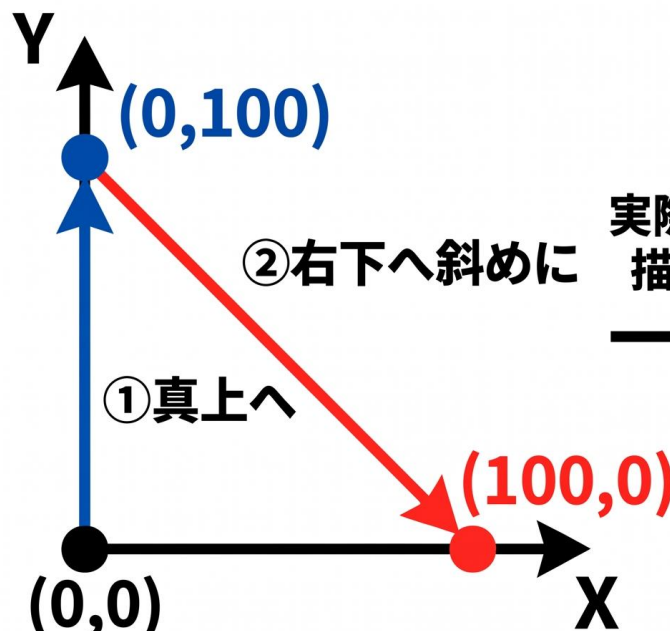
タートルグラフィックス



タートル（亀; turtle）が線を引きながら進む

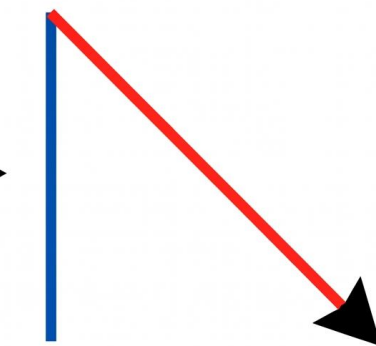
```
import turtle
t = turtle.Turtle()
t.goto(0,100)
t.goto(100,0)
```

《Python プログラム》



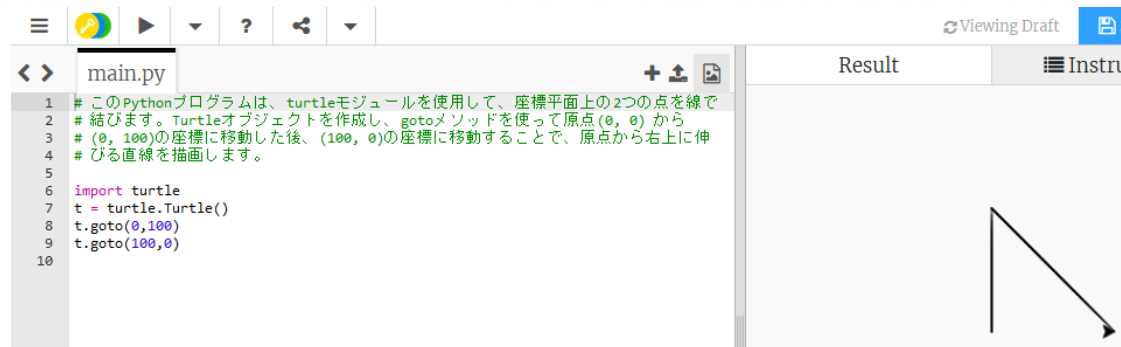
開始位置 = 画面中央

実際の
描画



描かれる図形

`goto(x,y)` : 現在地から座標(x,y)まで線を引き移動する



オブジェクトとメソッド



```
import turtle
```

```
t=turtle.Turtle()
```

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

インポート

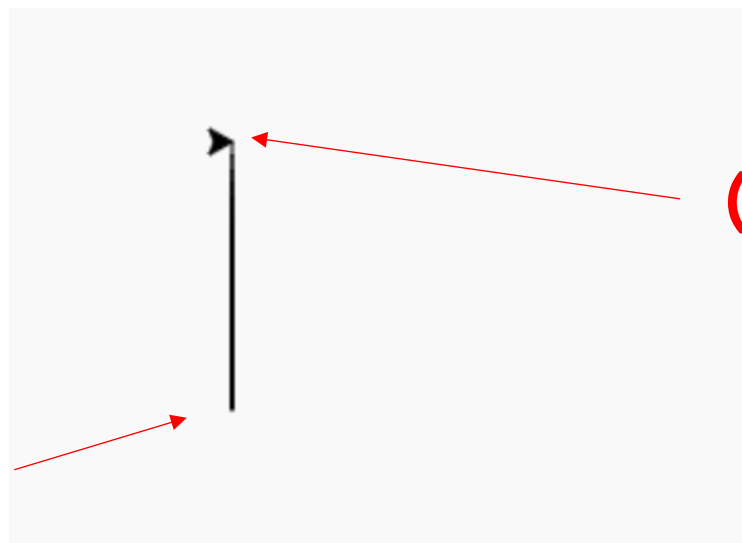
オブジェクト生成. tへのセット.

(0, 100)への移動

実行結果

**最初の位置は
(0, 0)**

(0, 0)



(0, 100)

オブジェクトが動く

オブジェクトとメソッド



```
import turtle
```

```
t=turtle.Turtle()
```

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

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

インポート

オブジェクト生成. tへのセット.

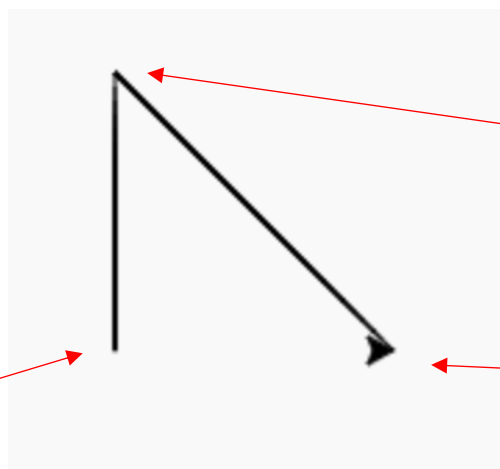
(0, 100)への移動

(100, 0)への移動

実行結果

**最初の位置は
(0, 0)**

(0, 0)



(0, 100)

(100, 0)

オブジェクトが動く

オブジェクト生成と代入



準備

```
import turtle
```

タートルグラフィックスの
機能を使う準備

オブジェクト生成

```
turtle.Turtle()
```

実体（オブジェクト）を
新しく1つ作る

変数に結びつける

```
t
```

実体

```
t = turtle.Turtle()
```

名札（変数）

実体を名札に結びつける（代入）

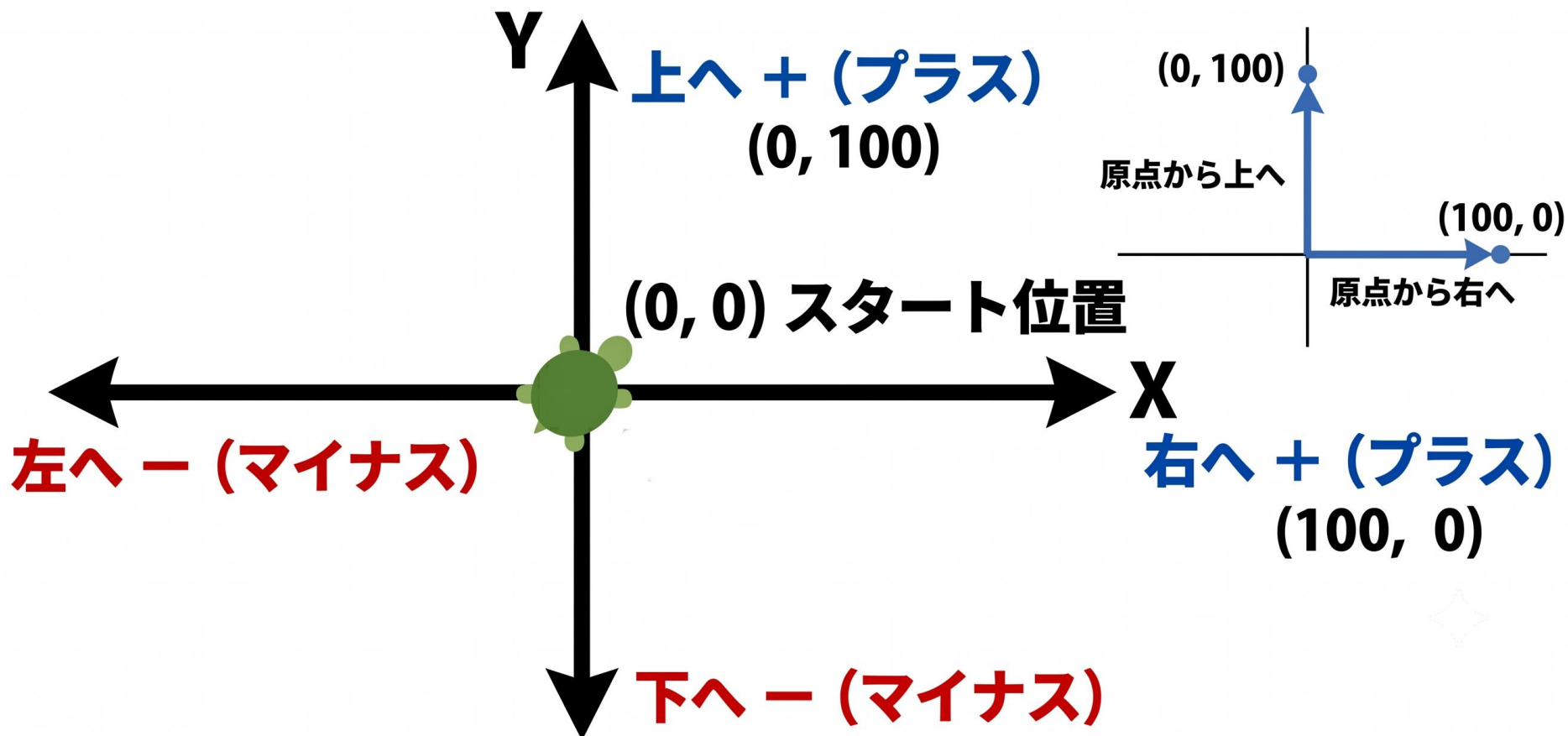
生成された

右辺で作った実体を、左辺の変数 t にセットしている

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

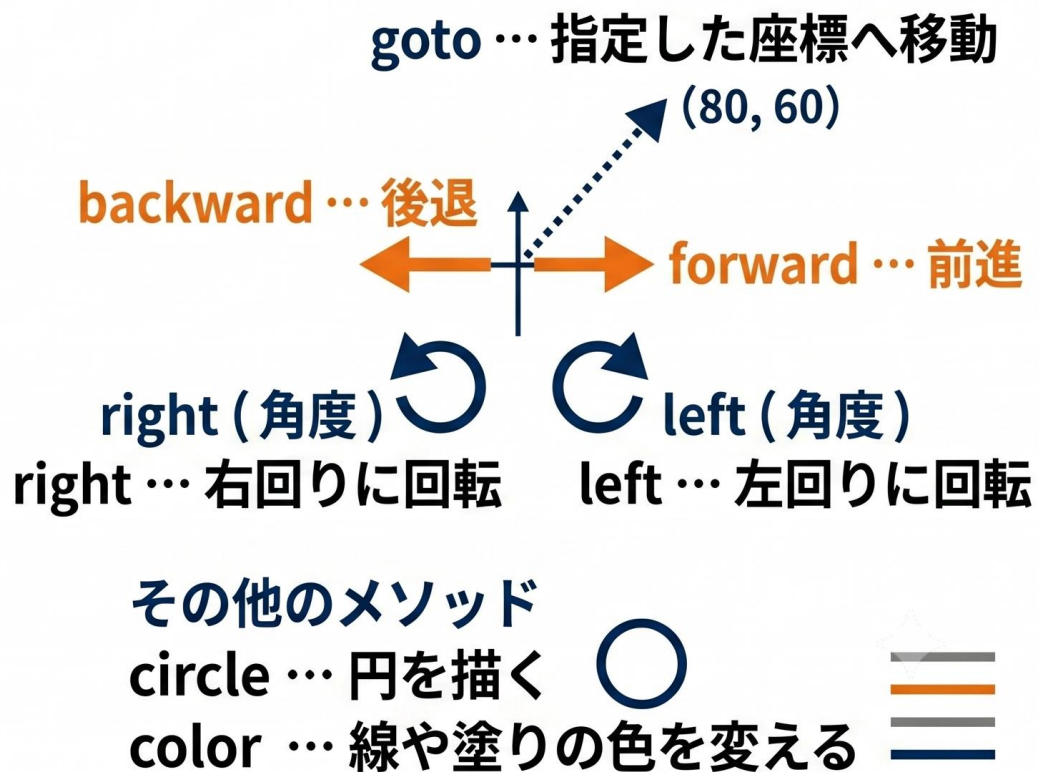


オブジェクトメソッドの関係

```
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



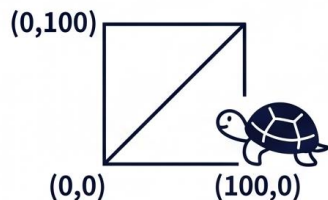
タートルグラフィックスで伸びる実力



turtleで図形を描く演習を通して、3つの学習姿勢を育てる

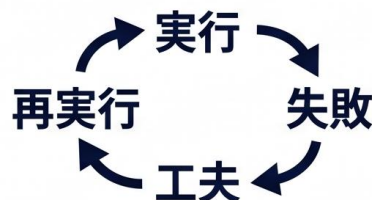
発想力・創造力

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



実力が育つ過程



演習



Trinket の概要



Trinket : ブラウザだけでPythonを書き、動かし、URLで公開・共有できる学習サイト

ブラウザ上で動く

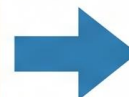
```
1 age = 18
2 if age <= 11:
3     print(500)
4 else:
5     print(1800)
6
```

公開する



自作プログラムを
公開

URLが
割り当て
られる



trinket.io/python/xxxx

インストール不要・ブラウザで完結

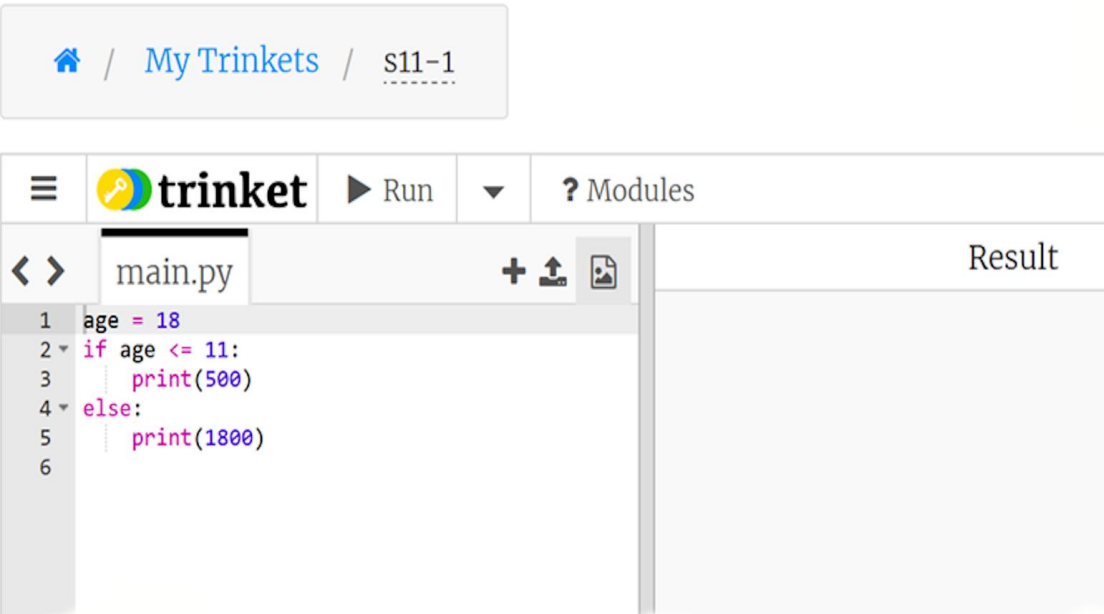
他の人と共有できる

公開ごとに固有のURL

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



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

① ここで編集して再実行できる

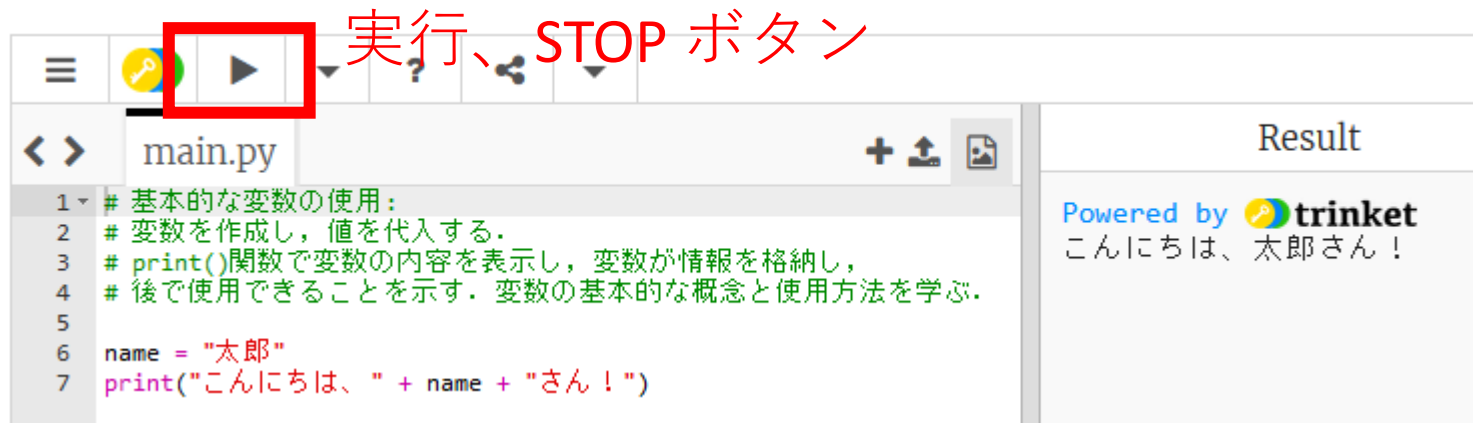
② 結果はここで確認する

演習 1. 変数と代入, + を用いた文字列連結

① trinket の次のページを開く

<https://trinket.io/python/abafd851480a>


② 実行結果が, 次のように表示されることを確認



実行、STOP ボタン

```
1 # 基本的な変数の使用:
2 # 変数を作成し, 値を代入する.
3 # print()関数で変数の内容を表示し, 変数が情報を格納し,
4 # 後で使用できることを示す. 変数の基本的な概念と使用方法を学ぶ.
5
6 name = "太郎"
7 print("こんにちは, " + name + "さん!")
```

Result

Powered by  trinket
こんにちは, 太郎さん!

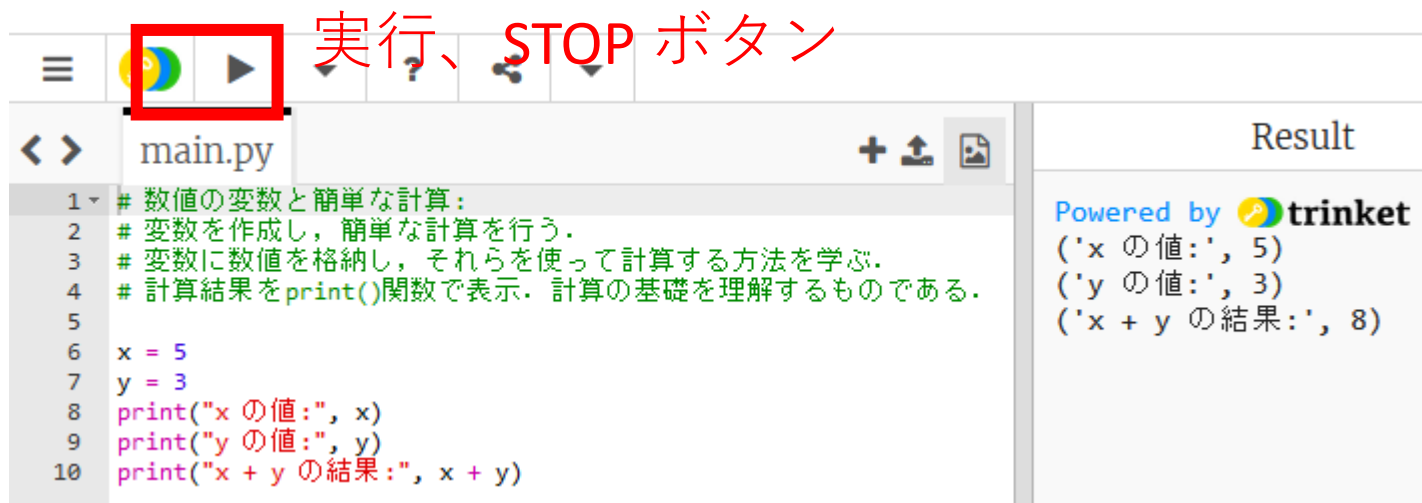
- 実行が開始しないときは、「**実行ボタン**」で**実行**

演習 2. 計算

① trinket の次のページを開く

<https://trinket.io/python/9870e86d63b9>

② 実行結果が、次のように表示されることを確認



実行、STOP ボタン

```
1 # 数値の変数と簡単な計算:  
2 # 変数を作成し, 簡単な計算を行う.  
3 # 変数に数値を格納し, それらを使って計算する方法を学ぶ.  
4 # 計算結果をprint()関数で表示. 計算の基礎を理解するものである.  
5  
6 x = 5  
7 y = 3  
8 print("x の値:", x)  
9 print("y の値:", y)  
10 print("x + y の結果:", x + y)
```

Result

Powered by trinket
('x の値:', 5)
('y の値:', 3)
('x + y の結果:', 8)

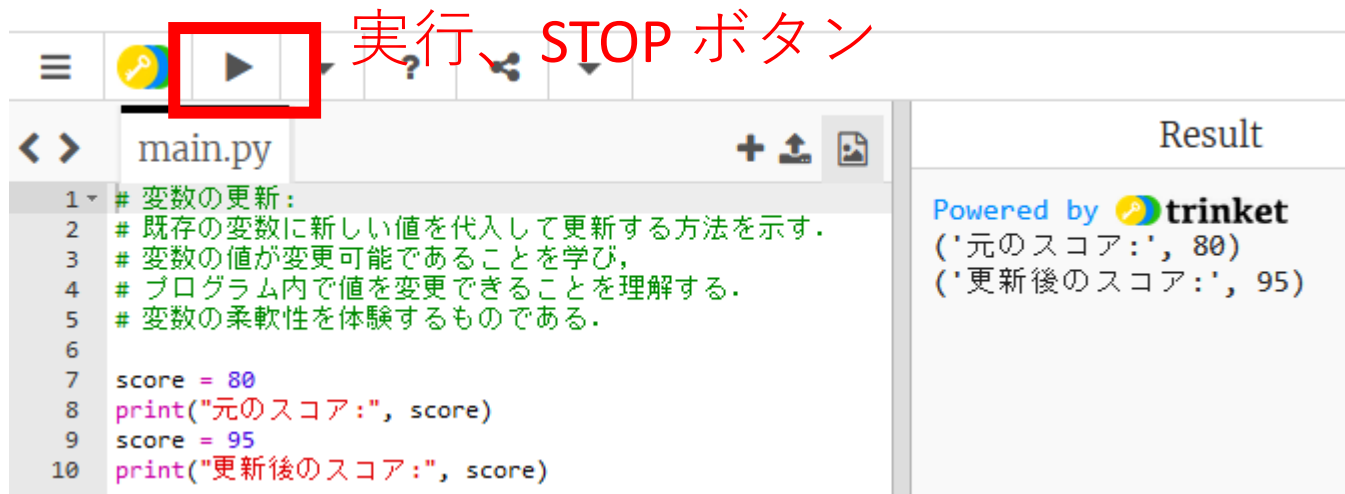
- 実行が開始しないときは、「**実行ボタン**」で**実行**

演習 3. 代入

① trinket の次のページを開く

<https://trinket.io/python/b869619b0874>


② 実行結果が、次のように表示されることを確認



実行、STOP ボタン

```
main.py
1 # 変数の更新:
2 # 既存の変数に新しい値を代入して更新する方法を示す.
3 # 変数の値が変更可能であることを学び,
4 # プログラム内で値を変更できることを理解する.
5 # 変数の柔軟性を体験するものである.
6
7 score = 80
8 print("元のスコア:", score)
9 score = 95
10 print("更新後のスコア:", score)
```

Result

Powered by  trinket
('元のスコア:', 80)
('更新後のスコア:', 95)

• 実行が開始しないときは、「実行ボタン」で実行

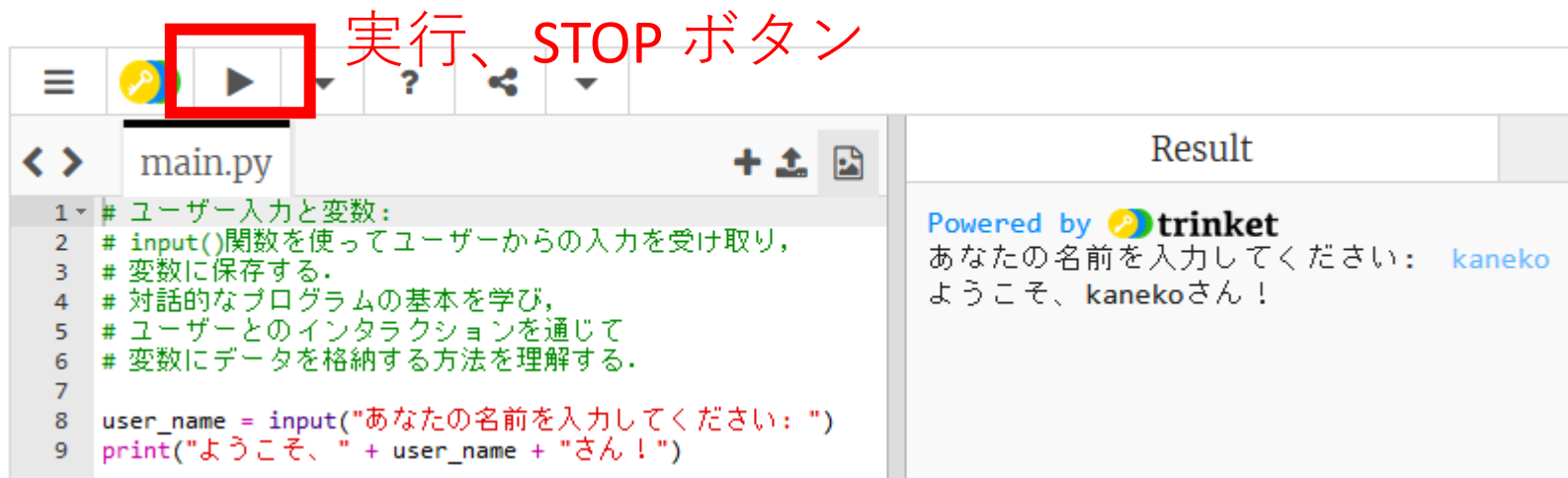
演習 4. input() による入力

① trinket の次のページを開く

<https://trinket.io/python/45f0bed92360>


② 実行結果が，次のように表示されることを確認

実行、STOP ボタン



```
1 # ユーザー入力と変数:  
2 # input()関数を使ってユーザーからの入力を受け取り,  
3 # 変数に保存する.  
4 # 対話的なプログラムの基本を学び,  
5 # ユーザーとのインタラクションを通じて  
6 # 変数にデータを格納する方法を理解する.  
7  
8 user_name = input("あなたの名前を入力してください: ")  
9 print("ようこそ、" + user_name + "さん!")
```

Result

Powered by  trinket
あなたの名前を入力してください: kaneko
ようこそ、kanekoさん!

• 実行が開始しないときは、「**実行ボタン**」で**実行**

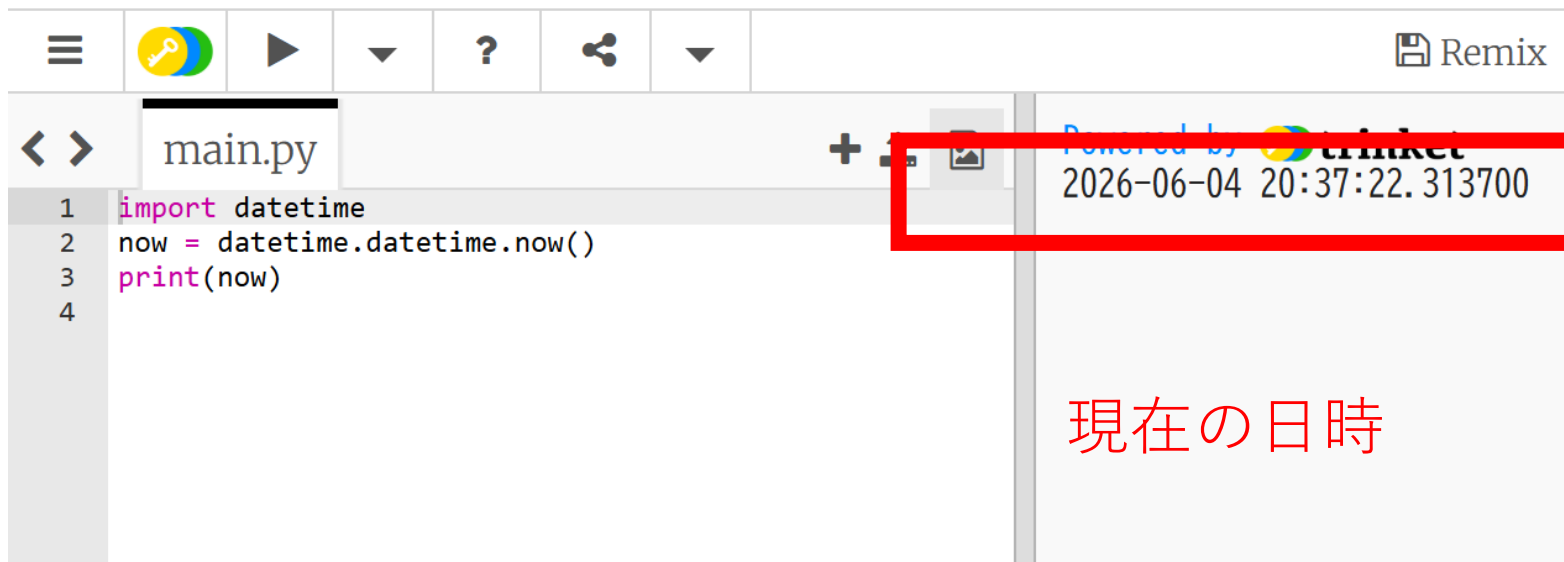
演習 5. オペレーティングシステム (コンピュータ) の タイマーを利用した**現在日時**の表示

① trinket の次のページを開く

<https://trinket.io/python/2b804ab19a>


② 実行結果が、次のように表示されることを確認

```
import datetime
now = datetime.datetime.now()
print(now)
```



main.py

```
1 import datetime
2 now = datetime.datetime.now()
3 print(now)
4
```

Powered by  trinket
2026-06-04 20:37:22.313700

現在の日時

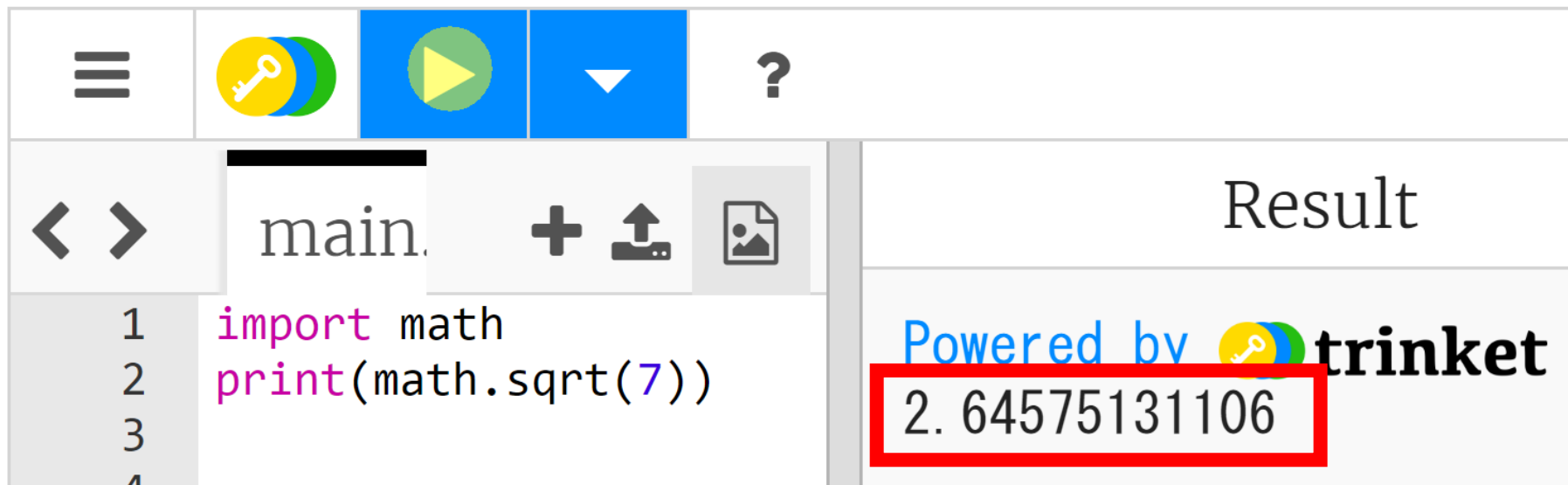
演習 6. 平方根：面積が 7 の正方形の一辺の長さ

① trinket の次のページを開く

<https://trinket.io/python/597e5771ff>

② 実行結果が、次のように表示されることを確認

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



The screenshot shows the Trinket Python IDE interface. The top bar contains a menu icon, a key icon, a play button, a dropdown arrow, and a question mark. Below the top bar, there are navigation arrows, a file name 'main.py', and icons for adding files, uploading, and inserting images. The code editor shows the following code:

```
1 import math
2 print(math.sqrt(7))
3
4
```

The output area on the right is titled 'Result' and displays the text 'Powered by trinket' with the key icon. Below this, the numerical result '2.64575131106' is shown and highlighted with a red rectangular box.

演習 7. 円周率：半径 3 の円の面積は？



⑤ trinket の次のページを開く

<https://trinket.io/python/4e3559f879>

⑥ 実行結果が、次のように表示されることを確認

```
import math
print(3 * 3 * math.pi)
```

The screenshot shows the Trinket Python IDE interface. The top toolbar contains a menu icon, a key icon, a play button, a dropdown arrow, and a question mark. Below the toolbar, the file name 'main.py' is displayed. The code editor shows the following Python code:

```
1 import math
2 print(3 * 3 * math.pi)
3
4
```

To the right of the code editor, the 'Result' section displays the output of the code execution: '28. 2743338823'. The output is highlighted with a red rectangular box. The text 'Powered by trinket' is visible above the output.



演習 8. 三角関数：三角形の2辺の長さが、**4**と**6**で、その間の角度が**60**度のとき、面積は $(1/2) \times 4 \times 6 \times \sin(60)$

⑦ trinket の次のページを開く

<https://trinket.io/python/bdcce27488>

⑧ 実行結果が、次のように表示されることを確認

```
import math
print((1/2) * 4 * 6 * math.sin(60 * math.pi / 180))
```

main.py

```
1 import math
2 print((1/2) * 4 * 6 * math.sin(60 * math.pi / 180))
3
4
```

Result

Powered by trinket

10.3923048454

演習 9. タートルグラフィックス



① trinketの次のページを開く

<https://trinket.io/python/f29bfe71cd>

② 実行結果が、次のように表示されることを確認

```
1 # このPythonプログラムは、turtleモジュールを使用して、座標平面上の2つの点を線で  
2 # 結びます。Turtleオブジェクトを作成し、gotoメソッドを使って原点(0, 0) から  
3 # (0, 100)の座標に移動した後、(100, 0)の座標に移動することで、原点から右上に伸  
4 # びる直線を描画します。  
5  
6 import turtle  
7 t = turtle.Turtle()  
8 t.goto(0,100)  
9 t.goto(100,0)  
10
```

• 実行が開始しないときは、「**実行ボタン**」で**実行**



③ 次のように書き換えて，再実行し，結果が変わることを確認

```
import turtle  
t = turtle.Turtle()  
t.goto(0,100)  
t.goto(200,0)
```

← 書き換え

実行，STOPボタン

```
main.py  
1 import turtle  
2 t = turtle.Turtle()  
3 t.goto(0,100)  
4 t.goto(100,0)  
5
```

Result

演習 10. タートルグラフィックスで六角形



- ① 別のプログラムを試す. trinketの次のページを開く

<https://trinket.io/python/ddb861147133>

- ② 実行結果が, 次のように表示されることを確認

実行, STOPボタン

```
main.py view the result.
1 # タートルグラフィックスを用いて六角形を描画する.
2 # forwardメソッドで50ピクセル前進し, leftメソッドで60度左回転する操作を
3 # 6回繰り返すことで, 正六角形を完成させる.
4
5 import turtle
6
7 t = turtle.Turtle()
8
9 # 六角形を描く
10 t.forward(50)
11 t.left(60)
12 t.forward(50)
13 t.left(60)
14 t.forward(50)
15 t.left(60)
16 t.forward(50)
17 t.left(60)
18 t.forward(50)
19 t.left(60)
20 t.forward(50)
21 t.left(60)
```

- 実行が開始しないときは, 「実行ボタン」で実行

演習 1 1. タートルグラフィックスで星形



<https://trinket.io/python/5366def2f4>

The screenshot shows a Python code editor with the following code:

```
1 # このPythonプログラムは、turtleモジュールを使用して、座標平面上の5つの点を線
2 # で結んだ星型の図形を描画します。Turtleオブジェクトを作成し、gotoメソッドを
3 # 使って指定された座標に順番に移動しながら線を引いていきます。最後に、始点の
4 # 座標に戻ることによって図形を完成させます。
5
6 import turtle
7 t = turtle.Turtle()
8 t.goto(0, 100)
9 t.goto(58, -80)
10 t.goto(-95, 30)
11 t.goto(95, 30)
12 t.goto(-58, -80)
13 t.goto(0, 100)
14
15
16
17
```

The output on the right shows a black five-pointed star with a small arrowhead at the top vertex.

各自の自発的な演習，自己研鑽の時間

■ 発想力，創造力

あなた自身がデザインした図形を描く。