


CS-12. 変数, 入力と出力, 関数

(コンピューターサイエンス)

URL: <https://www.kkaneko.jp/cc/cs/index.html>

金子邦彦



- 
- ① 実践的な学習。**Trinket と Python Tutor を活用。** 段階的に確認しながら学習
 - ② プログラミング基礎。**変数、入出力、関数、繰り返し、条件分岐**について、Pythonのプログラム例で理解を深める。**抽象化と関数の重要性**
 - ③ **ステップ実行によるプログラム動作の理解。** プログラムのエラー対処力と論理的思考力の向上
 - ④ Pythonプログラミングの基本、問題解決能力、自己学習力、実践的なツールの活用力の向上

アウトライン

1. for による繰り返し
2. 条件分岐のステップ実行
3. 入力と出力
4. 式の抽象化と関数
5. 関数定義, def

- **プログラミング**は**人間の力を増幅**し、私たちができることを大幅に広げる
- **プログラミング**はさまざまな分野で活用されている
 - シミュレーション：複雑な現象をモデル化し、予測や分析を行います
 - 大量データ処理：データの収集、加工、分析
 - AIシステムの開発
 - Webサイト, アプリケーションなどのソフトウェア
- **プログラミングはクリエイティブな行為**
- さまざまな**作業を自動化**したいとき、**問題解決**したいときにも役立つ

プログラミングの楽しさと達成感



• 楽しさ

- **未来の技術を学ぶことは楽しい。**
- **プログラミングは自分のアイデアを形にできるクリエイティブな行為。**
- **視覚的なプログラムを書くことで、ゲーム感覚をもって楽しみながら学習することも可能。**

• 達成感

- **プログラミングを通じて問題解決のスキルを身につけることは、大きな達成感につながる**
- **新しいソフトウェアや技術を生み出す。**

- Trinket は**オンライン**の Python、HTML 等の**学習サイト**
- 有料の機能と無料の機能がある
- **自分が作成した Python プログラムを公開し、他の人に実行してもらうことが可能**（そのとき、書き替えて実行も可能）
- **Python の標準機能**を登載、その他、次のモジュールやパッケージがインストール済み

math, matplotlib.pyplot, numpy, operator, processing, pygal, random, re, string, time, turtle, urllib.request

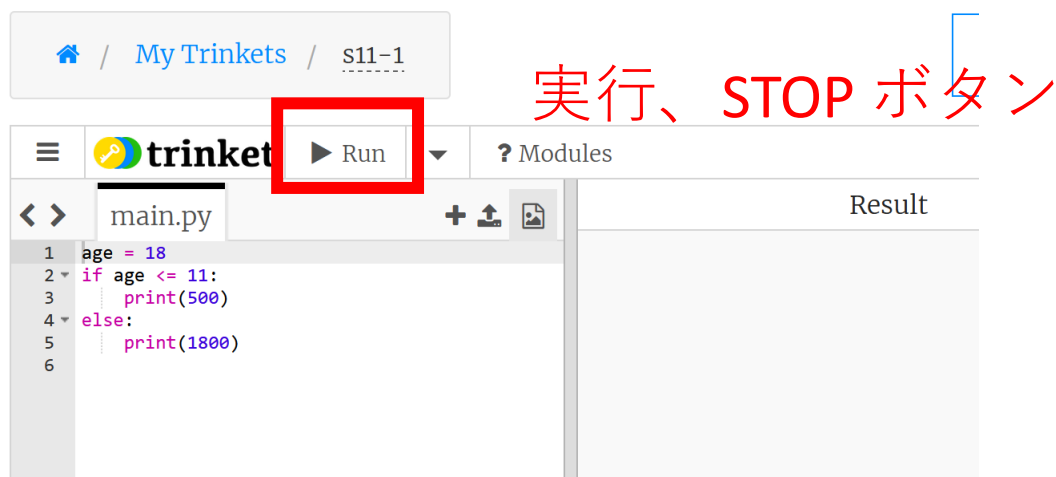


trinket でのプログラム実行

- 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/cdc4896571>

のように、違うプログラムには違う URL が割り当てられる



ソースコードの
メイン画面

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

12-1. for による繰り返し

タートルグラフィックス

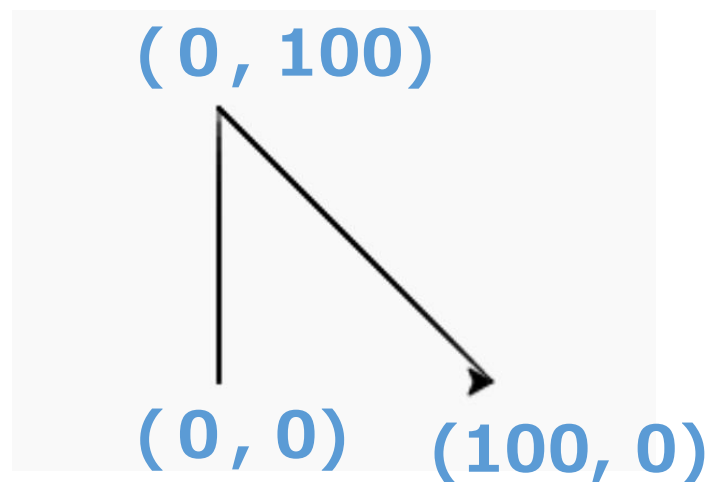


カーソルを使って絵を描く

- **タートルグラフィックス**を用いた演習により、**プログラムによって図形を描画**する。それを通して、プログラムの**動作を視覚的に理解**
- 論理的思考力や課題解決力の向上にもつながる

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

タートルグラフィックスの機能をインポートする「import turtle」が必要



タートルグラフィックス入門



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

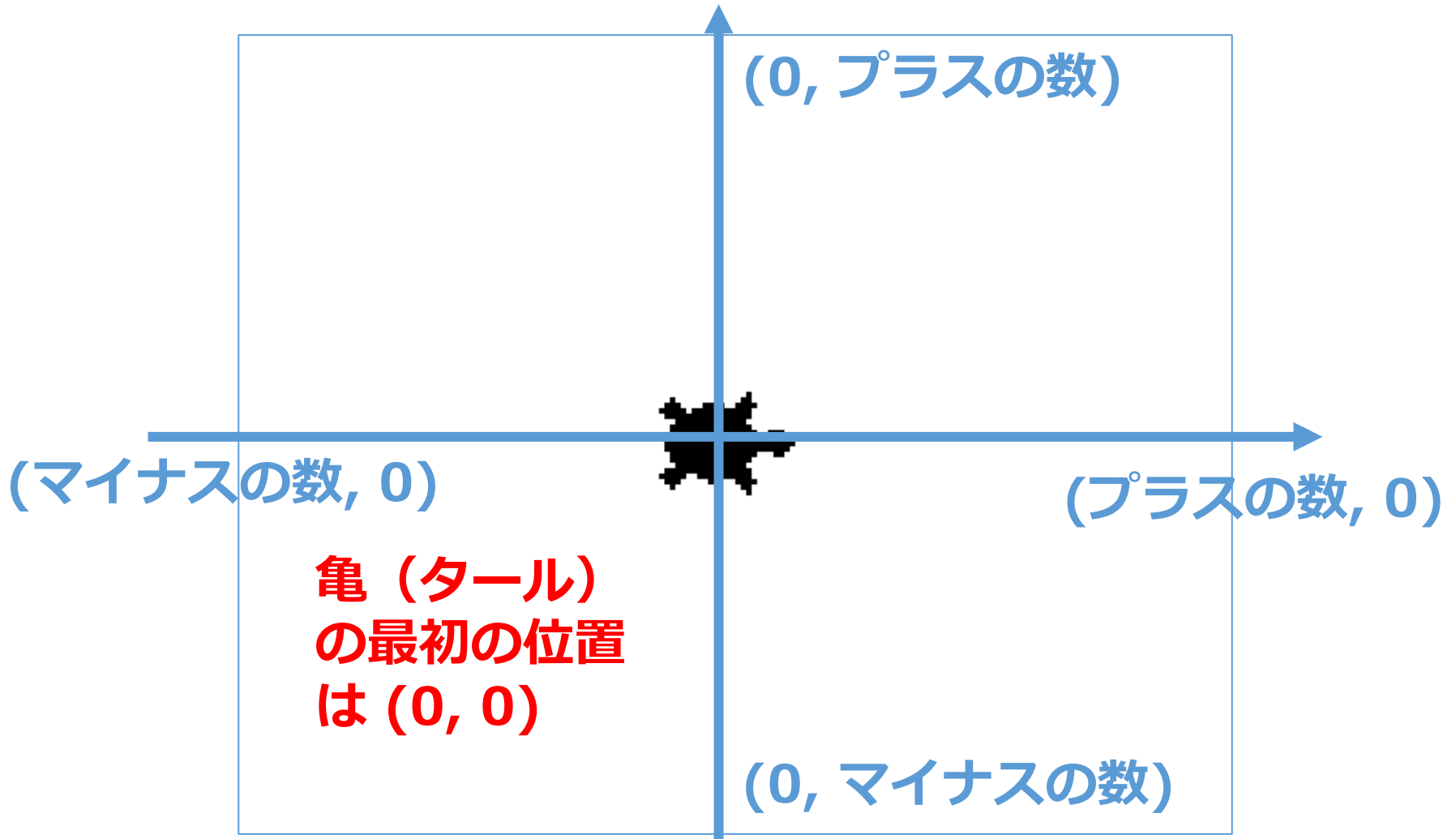
オブジェクト **メソッド**

- **メソッド**は、オブジェクトが持つ機能を呼び出すためのもの
- 「**goto**」は**指定した座標への移動**

主なメソッド

- **goto** (<横方向の値> , <縦方向の値>) **移動**
- **forward**(<移動量>) **前進**
- **backward**(<移動量>) **後退**
- **right**(<角度>) **右回りに回転**
- **left**(<角度>) **左回りに回転**

メソッド goto の引数となる 縦方向の値と、横方向の値



演習 1

for による繰り返し

ページ 12 ~ 14

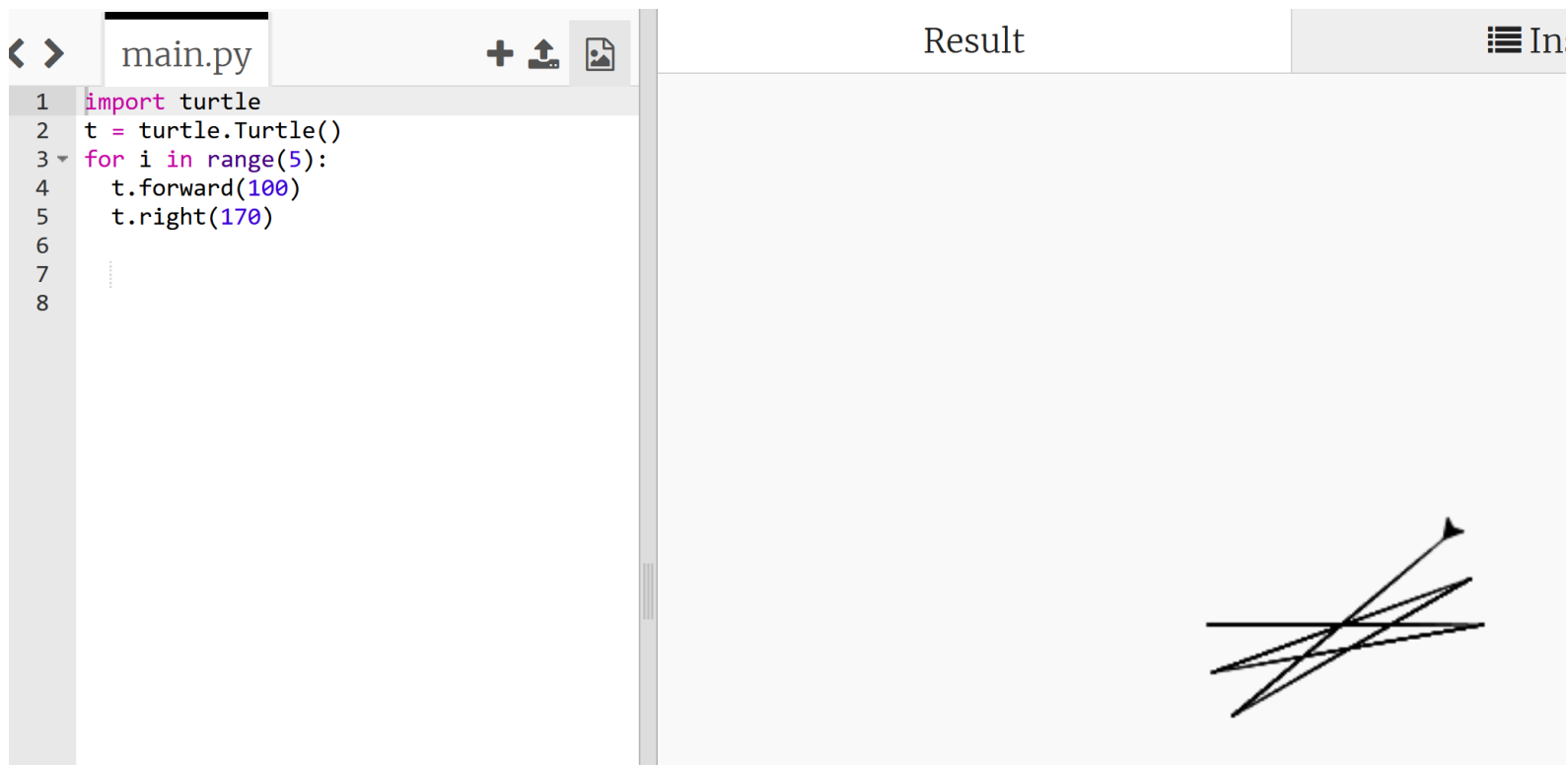
【トピックス】

- trinket の利用
- for による繰り返し

① trinket 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



The screenshot shows a code editor with a file named 'main.py'. The code is as follows:

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

The right side of the editor shows the 'Result' tab, which displays a drawing of a five-pointed star. The star is formed by five overlapping lines that intersect at a central point. Each line is 100 units long and is rotated 170 degrees from the previous one.

③ trinket 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



```
main.py
1 import turtle
2 t = turtle.Turtle()
3 for i in range(5):
4     t.forward(i * 20 + 100)
5     t.right(170)
6
7
8
```

Result



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

まとめ



- **for** を用いて**特定の処理を繰り返す**ことができる。
- 「for i in range(10)」は**0から9までの値を i に順次代入**する
- **タートルグラフィックスはカーソルで絵を描く**
ツールで、goto、forward、backward、right、left
などのメソッドで操作

12-2. 条件分岐のステップ実行

ステップ実行



- **ステップ実行**では、**1行ずつの実行**が行われ、そのときの変数の値の変化などを**確認**できる
- **ステップ実行**により、**プログラムの動作を細かく追跡**でき、不具合が発生している箇所の特定、プログラムの学習に役立つ
- **通常実行**は、**プログラムを最初から最後まで一度に実行**するもの（プログラム実行中の変数の値の変化を確認するなどは困難）。**ステップ実行**は、**プログラムを1行ずつ実行し、実行後にプログラムを一時停止**するもの。

条件分岐の Python プログラム



age の値が **11**以下 → **500**

12以上 → **1800**

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

条件式は「**age <= 11**」のようになる

Python Tutor

- Python Tutor というウェブサイトを利用しよう

<http://www.pythontutor.com/>

- Web ブラウザを使ってアクセスできる

- PythonTutor では, Pythonだけでなく, Java, C,, C++, JavaScript, Ruby など, 多くのプログラミング言語を学ぶことができる.



Python 3.6
[known limitations](#)

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

[Edit this code](#)

just executed
to execute

<< First < Prev Next >

Done running (16 s)

Trinket と Python Tutor の特徴と使い分け



Trinket

- 特徴：オンラインでPythonプログラムを実行・共有できる
- 使用目的：**基本的なプログラム実行と結果確認**

Python Tutor

- 特徴：プログラムの実行過程を視覚化し、変数の変化を追跡できる
- 使用目的：**プログラムの動作理解、デバッグ（プログラム中の誤りの発見と解決）スキルの向上**

Python Tutor の使用方法



- ① まず, **ウェブブラウザ**を開く
- ② **Python Tutor** を利用するために, 以下の URL にアクセス

<http://www.pythontutor.com/>

- ③ 「**Python**」 をクリック ⇒ **編集画面**が開く

Learn Python, JavaScript, C, C++, and Java

This tool helps you learn Python, JavaScript, C, C++, and Java programming by [visualizing code execution](#). You can use it to debug your homework assignments and as a supplement to online coding tutorials.

Start coding now in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

Over 15 million people in more than 180 countries have used Python Tutor to visualize over 200 million pieces of code. It is the most widely-used program visualization tool for computing education.

You can also embed these visualizations into any webpage. Here's an example showing recursion in Python:

Python Tutor の編集画面



Python debugger - [pdb](#) interface to Python Tutor - Learn Python by visualizing code (also debug [JavaScript](#), [Java](#), [C](#), and [C++](#) code)

Write code in 「Python 3.6」になっている

1 |

エディタ
(プログラムを書き換えることができる)

実行のためのボタン

[Show code examples](#)



Python Tutor でのプログラム実行

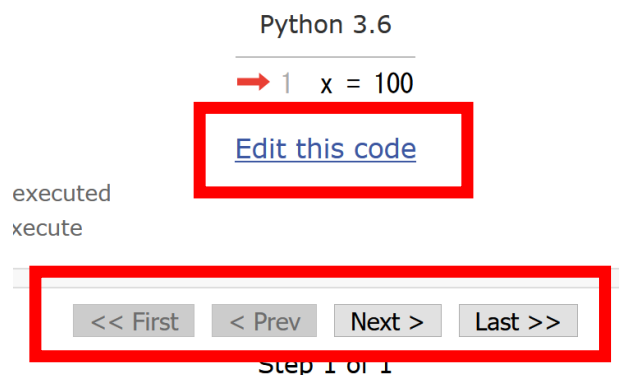
- Python Tutor は Python などのプログラムを書き実行できるサイト。ステップ実行、変数の値表示などの機能がある。
- Python Tutorのウェブサイトアクセス。「Python」を選択
<https://www.pythontutor.com/>

メイン画面で、プログラムを書く



Visualize Execution ボタン

メイン画面に
戻るには
Edit this code

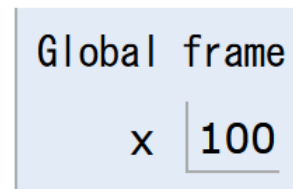


通常実行: Last

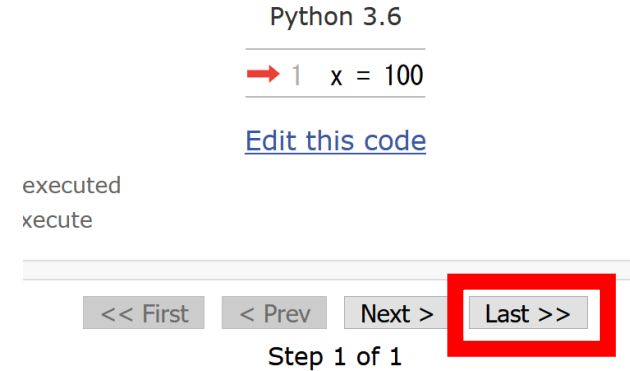
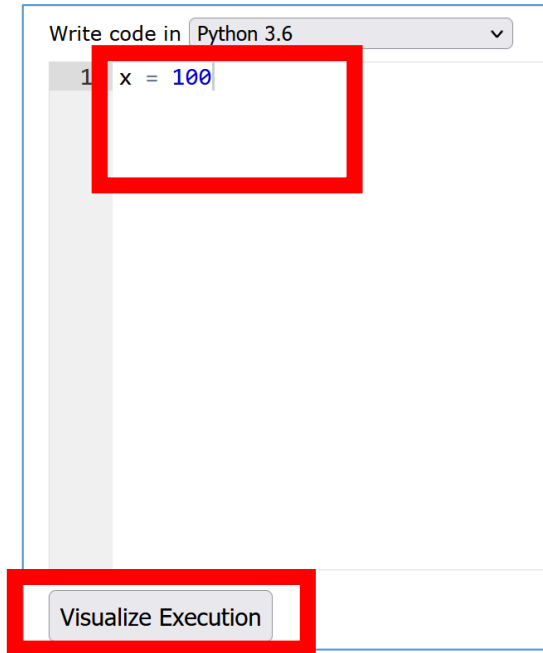
ステップ実行: First, Prev, Next

変数の値を
視覚的に
確認できる

Frames

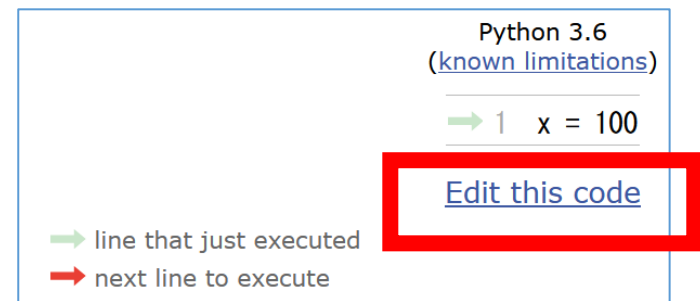
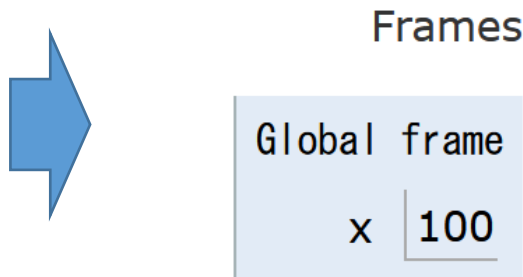


Python Tutor でのプログラム実行手順



(1) 「**Visualize Execution**」をクリックして**実行画面**に切り替える

(2) 「**Last**」をクリック。



(3) 実行結果を確認する。

(4) 「**Edit this code**」をクリックして**編集画面**に戻る

Python Tutor 使用上の注意点①



実行画面で、**赤いエラーメッセージ**が出ることがある

過去の文法ミスに関する確認表示。

基本的には、**無視して問題ない**

邪魔なときは「**Close**」

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

Python 3.6
([known limitations](#))

```
→ 1 x = 100
```

[Edit this code](#)

→ line that just executed
→ next line to execute

<< First < Prev Next > Last >>

Step 1 of 1

[Customize visualization](#)

Frames Objects

You just fixed the following error:

```
1 x = 100!
```

SyntaxError: invalid syntax (<string>, line 1)

Please help us improve this tool with your feedback.
What misunderstanding do you think caused this error?

Submit Close Hide all of these pop-ups

Python Tutor 使用上の注意点②



「please wait ... executing」のとき，10秒ほど待つ。



- Python Tutor が混雑しているとき，「Server Busy . . .」 と表示される場合がある。
- このメッセージは，サーバが混雑していることを示す。
- 数秒から数十秒待つと自動で処理が始まるはずですが（しかし，表示が変わらないときは，操作をもう一度試してください）

演習 2

条件分岐のステップ実行

資料：27～37

【トピックス】

- Python Tutor
- 字下げ
- :
- 条件分岐
- if
- else
- ステップ実行

ステップ実行により確認できること

ステップ実行により，ジャンプの様子を観察

ジャンプの様子を示す矢印

```
Python 3.6
1 age = 30
2 if age <= 12:
3     print(500)
4 else:
5     print(1200)
```

[Edit this code](#)

Print output (drag lower right)

Frames

Global frame

age 30

変数の値の確認もできる

Step 3 of 3

<< First < Prev Next > Last >>

ステップ実行は、ボタンやスライダーでコントロール

Python Tutor の起動



① **ウェブブラウザ**を起動する

② **Python Tutor** を使いたいので, 次の URL を開く
<https://www.pythontutor.com/>

③ 「**Python**」 をクリック ⇒ **メイン画面**が開く

Learn Python, JavaScript, C, C++, and Java

This tool helps you learn Python, JavaScript, C, C++, and Java programming by [visualizing code execution](#). You can use it to debug your homework assignments and as a supplement to online coding tutorials.

Start coding now in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

Over 15 million people in more than 180 countries have used Python Tutor to visualize over 200 million pieces of code. It is the most widely-used program visualization tool for computing education.

You can also embed these visualizations into any webpage. Here's an example showing recursion in Python:

④ Python Tutor のエディタで次のプログラムを入れる

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

if (age <= 11)の直後に「:」
else の直後に「:」
(どちらも、コロン)

字下げも正確に！

print の前に、「タブ (Tab)」を1つだけ

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

正しくない字下げ

「delキー」などを使い
ながら編集

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

正しい字下げ

⑤ 通常実行するために、「Visual Execution」をクリック. そして「Last」をクリック. 結果 1800を確認

Python 3.6
[known limitations](#)

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

[Edit this code](#)

Executed
Output



<< First < Prev Next > Last >>

Done running (3 steps)

Print output (drag lower right corner to

1800

Frames

Objects

Global frame

age 18

結果の
「1800」を確認



⑥プログラム実行を最初の行に戻す操作 「**First**」をクリックして、最初の行に戻す

Python 3.6
[known limitations](#)

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

[Edit this code](#)

executed
xecute

<< First

< Prev

Next >

Last >>

Done running (3 steps)

- ⑦ 「**Step 1 of 3**」と表示されているので、
全部で、**ステップ数**は**3**あることが分かる
(ステップ数と、プログラムの行数は**違うもの**)

Python 3.6
[known limitations](#)

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

[Edit this code](#)

ecuted
cute

<< First

< Prev

Next >

Last >>

Step 1 of 3

⑧ **ステップ**実行したいので、「Next」をクリックしながら、矢印の動きを確認。

※「Next」ボタンを何度かクリックし、それ以上進めなくなったら終了

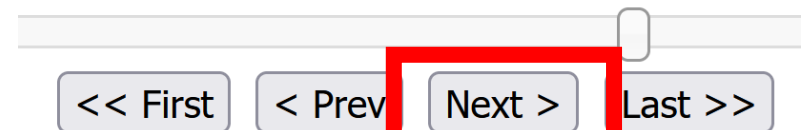
見どころ
2行目から 5行目へ
ジャンプするところ



```
Python 3.6  
known limitations  
-----  
1 age = 18  
→ 2 if age <= 11:  
3     print(500)  
4 else:  
→ 5     print(1800)
```

[Edit this code](#)

executed
acute



Step 3 of 3

⑨ First, Prev, Next, Last ボタンとスライダーによりプログラム実行を制御してみる

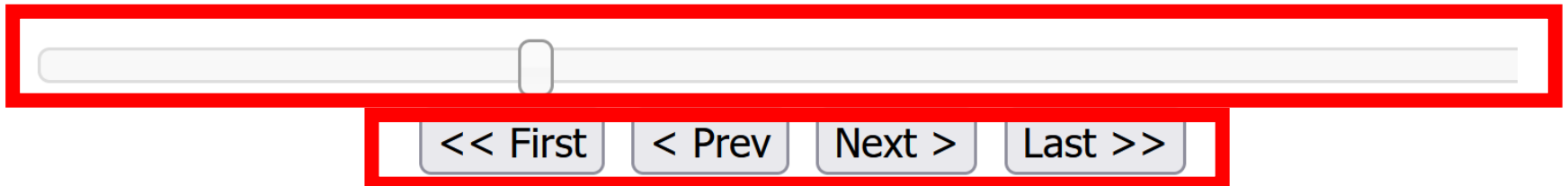
Python 3.6
[known limitations](#)

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

[Edit this code](#)

→ line that just executed

→ next line to execute



The control panel consists of a horizontal slider at the top with a small white knob. Below the slider are four buttons: '<< First', '< Prev', 'Next >', and 'Last >>'. The entire panel is enclosed in a thick red border.

Step 2 of 3



⑩ メイン画面に戻るには、「**Edit this code**」をクリック

Python 3.6
[known limitations](#)

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

[Edit this code](#)

: executed
execute

<< First < Prev Next > Last >>

Done running (3 steps)

ステップ実行 まとめ



- **通常実行**は、プログラムを最初から最後まで一度に実行する
- **ステップ実行**は、プログラムを1行ずつ実行し、実行後にプログラムを一時停止するもの
- **ステップ実行**により、プログラムの動作を細かく追跡でき、不具合が発生している箇所の特特定、プログラムの学習に役立つ

12-3. 入力と出力

入力と出力



- 入力は、他のコンピュータや人間などが、コンピュータにデータを入れる
 - **input** は、**キーボードから与えられたデータ（文字列）**を、**Enter** キーが押されるまで読み込む。
- 出力は、コンピュータが、他のコンピュータや人間などにデータを出す
 - **print** は、**メッセージ（文字列）** や、**変数の値の表示** を行う

演習 3

input による入力と print による出力

ページ 4 1 ~ 4 3

【トピックス】


- trinket の利用
- input
- print

① trinket の次のページを開く

<https://trinket.io/python/bdca234a3e>

② 実行する。

③ 右下の画面で **3 Enter キー**


```
Powered by  trinket
teiken =
3
```

```
1 print("teiken =")
2 teihen = float(input())
3 print("takasa =")
4 takasa = float(input())
5 print("teihen * takasa / 2 =", teihen * takasa / 2)
```

④ 右下の画面で, 続けて **5 Enter キー**

```
Powered by  trinket
teiken =
3
takasa =
5
```

⑤ 結果の **7.5** を確認

```
Powered by  trinket
teiken =
3
takasa =
5
('teihen * takasa / 2 =', 7.5)
```

⑥ **3角形の面積を求めるプログラム**である. いろいろ試してみよう.




① trinket の次のページを開く

<https://trinket.io/python/3b490869e4>


② 実行する。

③ 右下の画面で **3 Enter キー**

```
1 print("r =")
2 r = float(input())
3 print("r * r * 3.14 =", r * r * 3.14)
```

```
Powered by  trinket
r =
3
```

④ 結果の **28.26** を確認

```
Powered by  trinket
r =
3
('r * r * 3.14 =', 28.26)
```

⑤ 円周率を 3.14 として、半径から円の面積を求めるプログラムである。いろいろ試してみよう。

12-4. 式の抽象化と関数

プログラミングの本質を考える



以下の問いについて、あなたの考えを書き出してみましよう。

1. プログラミングで最も難しい課題は何だと思えますか？

例: バグの少ないプログラムを書くこと、複雑な問題を解決すること

2. 効率的にプログラムを書くために必要な基本スキルは何でしょうか？

例: 論理的思考力、問題分析力

これらの問いについて考えたら、次のページから学ぶ「抽象化」という概念が、あなたの回答とどのように関連しているか注目しながら読み進めてください

プログラミングでの抽象化



- プログラミングでの根本問題：
誤り（バグ）のないプログラムの作成
- プログラミングの一番の基礎
抽象化を行うこと

以降のページでは、**抽象化**の概念とその**重要性**について詳しく説明します。

抽象化：複雑さを単純化する技術



抽象化は、複雑な詳細を隠し、本質的な特徴だけを取り出す

例：交通手段の抽象化

タクシー、バス、鉄道、自転車、徒歩

抽象化後：「移動手段」

共通の本質：ある場所から別の場所へ移動すること

プログラミングでの応用：

抽象化を使うと、複雑な問題を扱いやすくすることができる

プログラミングにおける抽象化



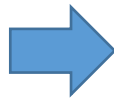
式の抽象化：類似した複数の式を1つにまとめる

複数の式

$$100 * 1.1$$

$$150 * 1.1$$

$$400 * 1.1$$



$$a * 1.1$$

抽象化された式

a は任意の数を表す **変数**

「ある値に 1.1 を掛ける」
という共通の操作を行う

関数による抽象化

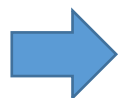


複数の式

100 * 1.1

150 * 1.1

400 * 1.1



a * 1.1



```
def foo(a):  
    return a * 1.1
```

抽象化された式

抽象化された式を
関数として定義

《利点》

抽象化された式を関数として定義することで、

- 同じ式を**何度も書く必要がなくなる**
- 「抽象化された式」に**名前**が付くので、分かりやすくなる
- 変更は、関数内の変更だけで済む

関数による抽象化：プログラムの再利用性と可読性の向上



①コードの簡潔化

抽象化前:

```
price1 = 100 * 1.1
```

```
price2 = 150 * 1.1
```

```
price3 = 400 * 1.1
```

抽象化後

```
def foo(a):
```

```
    return a * 1.1
```

```
price1 = foo(100)
```

```
price2 = foo(150)
```

```
price3 = foo(400)
```

関数による抽象化：プログラムの再利用性と可読性の向上



抽象化前:

```
price1 = 100 * 1.1  
price2 = 150 * 1.1  
price3 = 400 * 1.1
```

抽象化後

```
def foo(a):  
    return a * 1.1  
price1 = foo(100)  
price2 = foo(150)  
price3 = foo(400)
```

② エラーリスクの低減

- 計算式を1回だけ書くため、タイプミスなどのヒューマンエラーが減少

③ 保守性の向上

例：税率が10%から8%に変更された場合 修正箇所は1か所だけ：

```
def foo(a):  
    return a * 1.08
```

抽象化の重要性：40ページの問いかけへの 詳細な回答



- プログラミングでの根本問題

誤り（バグ）のないプログラムの作成

抽象化により、同じようなことを何度も書くことが減り、エラーの可能性が減少

- プログラミングの一番の基礎

抽象化を行うこと

式の抽象化により、バグの防止ができ、プログラムの変更が容易になる

抽象化はプログラミングの根本問題を解決し、基礎となる重要な概念です。

12-5. 関数定義, def

```
def foo(a):  
    return a * 1.1
```

- この**関数の本体**は
「`return a * 1.1`」
- この**関数**は、式「`a * 1.1`」に、名前 `foo` を付けたものと考えることもできる

式の抽象化と関数



抽象化前

```
1 print(100 * 1.1)
2 print(150 * 1.1)
3 print(200 * 1.1)
```

類似した複数の**式**

Powered by  **trinket**

```
110.0
165.0
220.0
```

実行結果

抽象化後

```
1 def foo(a):
2     return a * 1.1
3 print(foo(100))
4 print(foo(150))
5 print(foo(400))
```

関数の定義と使用

Powered by  **trinket**

```
110.0
165.0
440.0
```

同じ
実行結果になる

演習 4

関数を定義し使ってみる
ページ 56, 57

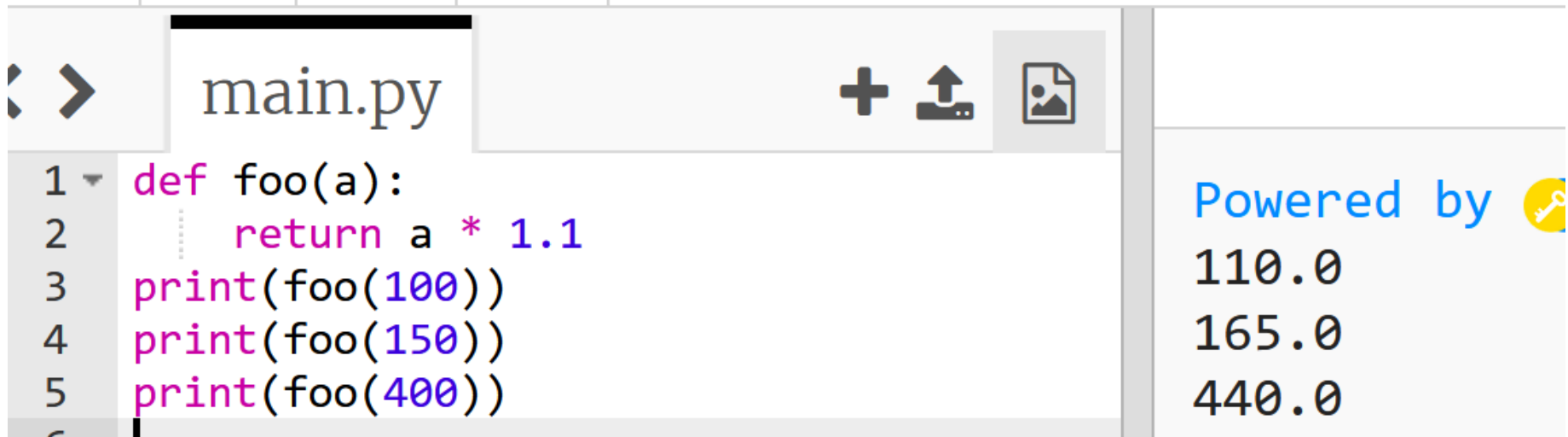
【トピックス】

- trinket の利用
- 式の抽象化と関数
- 関数定義
- def

① trinket の次のページを開く

<https://trinket.io/python/68a090babf>

② 実行結果が、次のように表示されることを確認



The screenshot shows a code editor with a file named 'main.py'. The code defines a function 'foo(a)' that returns 'a * 1.1' and prints the results for 'foo(100)', 'foo(150)', and 'foo(400)'. The execution output on the right shows 'Powered by' followed by the values '110.0', '165.0', and '440.0'.

```
main.py  
1 def foo(a):  
2     return a * 1.1  
3 print(foo(100))  
4 print(foo(150))  
5 print(foo(400))  
6  
Powered by  
110.0  
165.0  
440.0
```

全体まとめ

プログラム実行の方法

- **通常実行:** プログラムを最初から最後まで一度に実行
- **ステップ実行:** プログラムを1行ずつ実行し、動作を細かく追跡

Python プログラムの基本構造

- **繰り返し:** 特定の処理を反復
- **条件分岐:** プログラムの流れを制御
- **入出力:** `input()`で入力、`print()`で表示

抽象化と関数

- **式の抽象化:** 類似した複数の式を一つにまとめる
- **関数定義:** `def` キーワードを使用して式に名前をつける

例: `def foo(a):`

```
    return a * 1.1
```

- 抽象化と関数の利点: 同じ処理を何度も書く必要がなくなる、可読性の向上、保守性の向上: 変更が1箇所済み、バグの防止にも貢献

学習ツール

- **Trinket:** オンラインでのプログラム実行と結果確認
- **Python Tutor:** プログラムの実行過程の視覚化とデバッグ

今回の授業の学ぶ意義と満足感

- ① **実践的スキルと思考力の向上。** Pythonプログラミングの基礎習得。問題解決能力の向上。「**抽象化**」の理解。論理的思考力の向上。
- ② 学習体験。**プログラム実行の成功体験。** 実践的ツールによる体験。抽象化と関数の有用性の実感。
- ③ デジタル社会のための実力。**将来のプログラミング応用へ向けた基礎取得。** 継続的学習の基礎構築。
- ④ **実用性と将来性。** プログラミングは、作業の自動化やデータ分析などの多様な用途がある。将来のITエンジニアとしての実力強化。