pf-8. 式の抽象化と関数

（Python 入門）

URL: https://www.kkaneko.jp/pro/pf/index.html

金子邦彦

① プログラムの中のバグの解決は面倒である

② プログラムの修正時にバグは増えるということはよくある

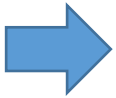
式の抽象化

100 * 1.1

150 * 1.1

400 * 1.1

類似した複数の**式**



a * 1.1

変数 a を使って，複数の**式**を１つにまとめる（**抽象化**）

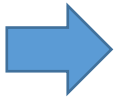
式の抽象化と関数

100 * 1.1

150 * 1.1

400 * 1.1

類似した複数の**式**



a * 1.1

変数 a を使って，複数の**式**を１つにまとめる（**抽象化**）

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

式「**a * 1.1**」を含む**関数 foo** を**定義**

関数 foo を使用.
100, 150, 400 は**引数**

Print output (drag lower righ

```
110.00000000000001
165.0
440.00000000000006
```

抽象化がなぜ大切なのか

- プログラミングでの根本問題は何でしょうか？
  誤り（バグ）の無いプログラムの作成

- プログラミングの一番の基礎は何でしょうか？
  - **抽象化を行うこと.**
  - **式の抽象化**により冗長な記述を避けることができ，**バグの防止やプログラムの変更が容易に**なる.

関数定義

```
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

trinket

- **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）

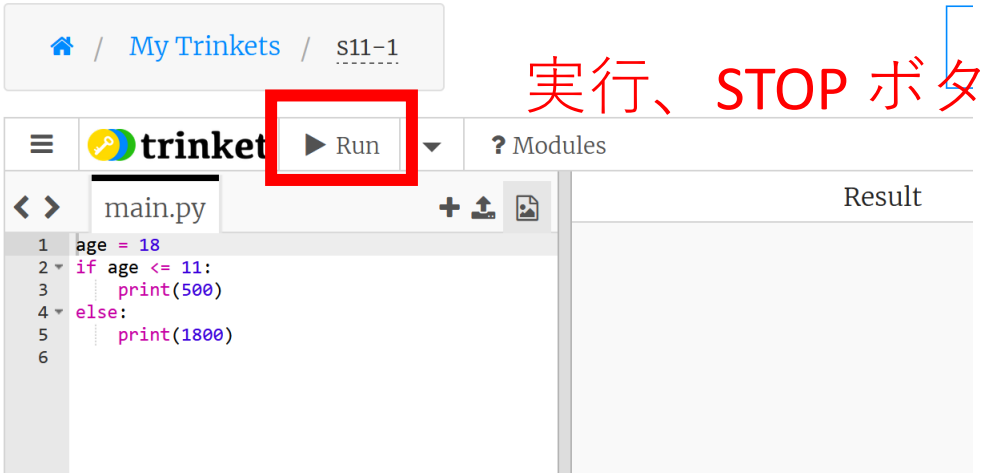
- **Python の標準機能**を登載、その他、次のパッケージがインストール済み

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/0fd59392c8

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

演習

関数を定義し使ってみる
ページ１１

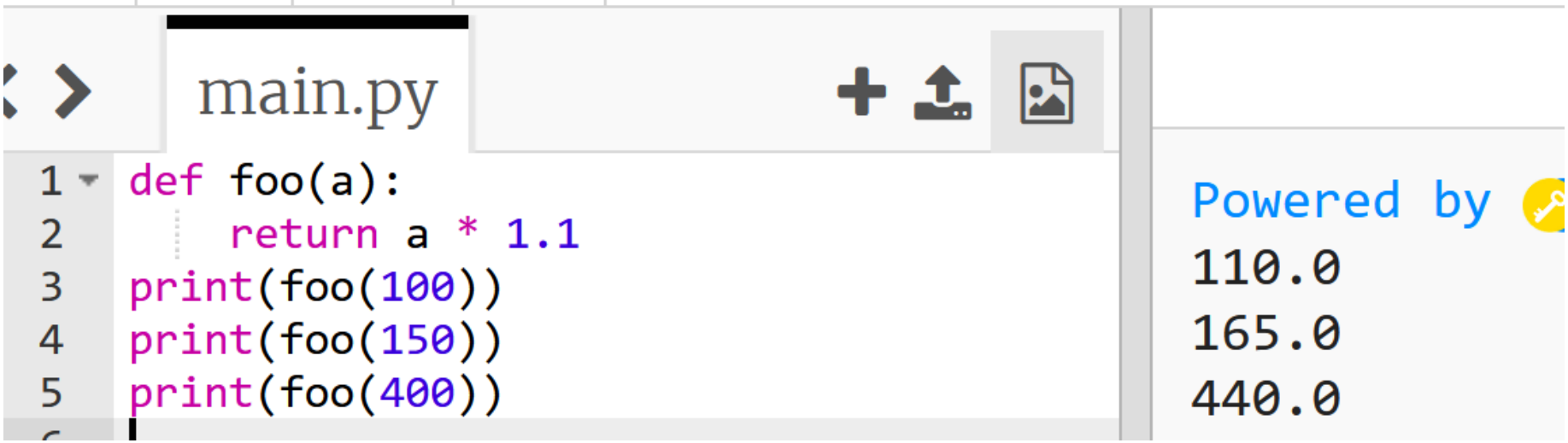
【トピックス】

- **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



① プログラムの中のバグの解決は面倒である

バグ解決と防止には抽象化と関数が不可欠です。関数は**特定の処理の部分を抜き出したもの**と考えることもできます。**抽象化と関数により、同じプログラムを繰り返し書く必要が減り**、**エラーの原因となる可能性を低減**し，全体のコードの品質を高めることができます。

② プログラムの修正時にバグは増えるということはよくある

プログラムの修正は慎重に行うべきです．**関数や抽象化の活用はその作業を容易にします**。特定の処理の部分を関数として定義することで、その処理を何度も書く手間を省き、修正時のエラー可能性を減らします。その結果、バグの発生リスクを低減します。

全体まとめ

- **式の抽象化**は，**類似した複数の式を一つにまとめる**こと
- **式の抽象化**により冗長な記述を避けることができ，**バグの防止やプログラムの変更が容易に**なる．
- **関数**は**式に名前をつける方法**で，これによりプログラムの再利用性が向上する

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