
Raspberry Pi 3を用いた Pythonプログラミング

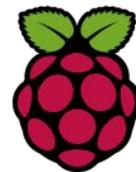
製作中の物です

Pythonとは？



1991年に開発されたプログラミング言語です
主にwebアプリ開発や、AI開発などに用いられていて
読みやすく、簡潔であることが設計思想であり、
初心者でも触れやすい言語であるとされています

Raspberry Piとは？



Raspberry Pi

2012/2/29に発売が開始された、

ARMプロセッサを搭載している、

シングルボードコンピューターという分類のものです

簡単にいうなら手のひらサイズのパソコンです

日本では略して、ラズパイと呼ばれていることが多いです

簡潔なまとめ

Python

→比較的簡単なプログラミング言語

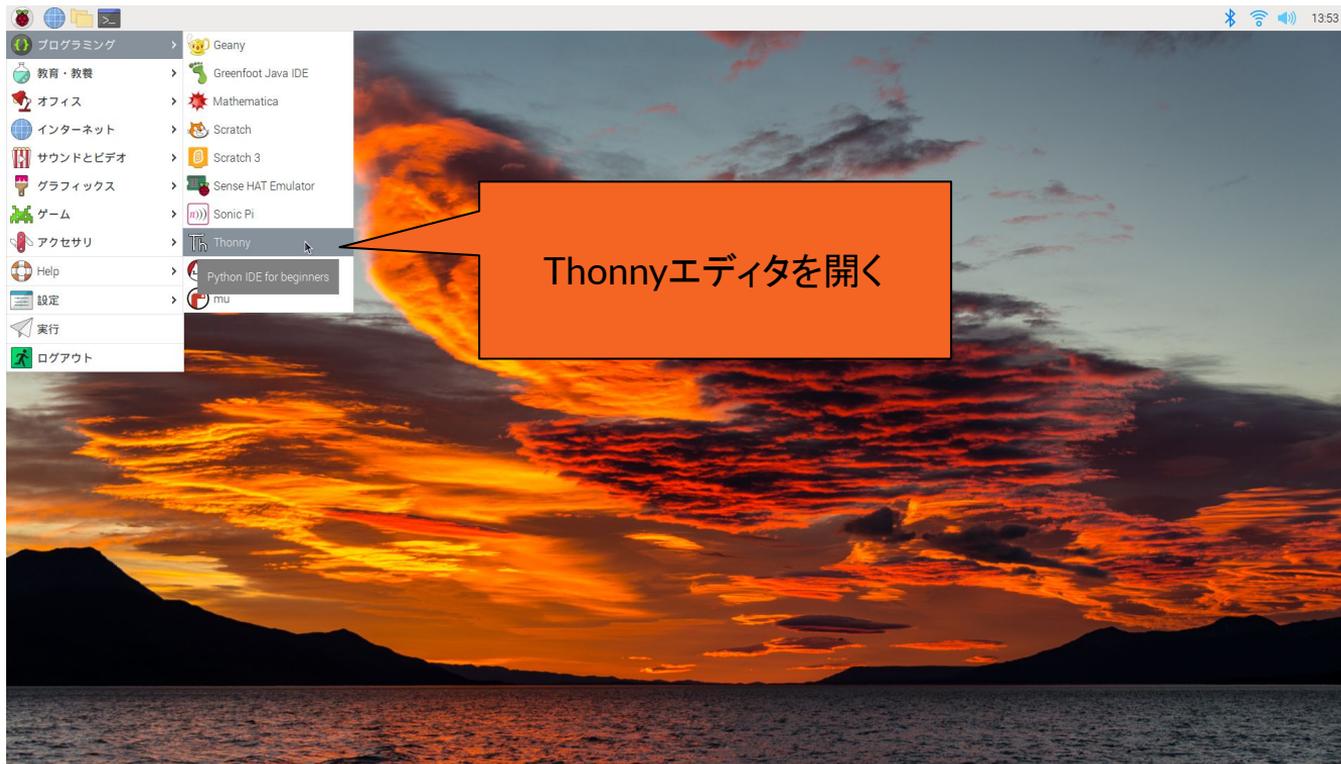
Raspberry Pi

→ARMプロセッサを搭載した

シングルボードコンピューター

手のひらサイズのパソコン

Pythonプログラミングを始めてみよう

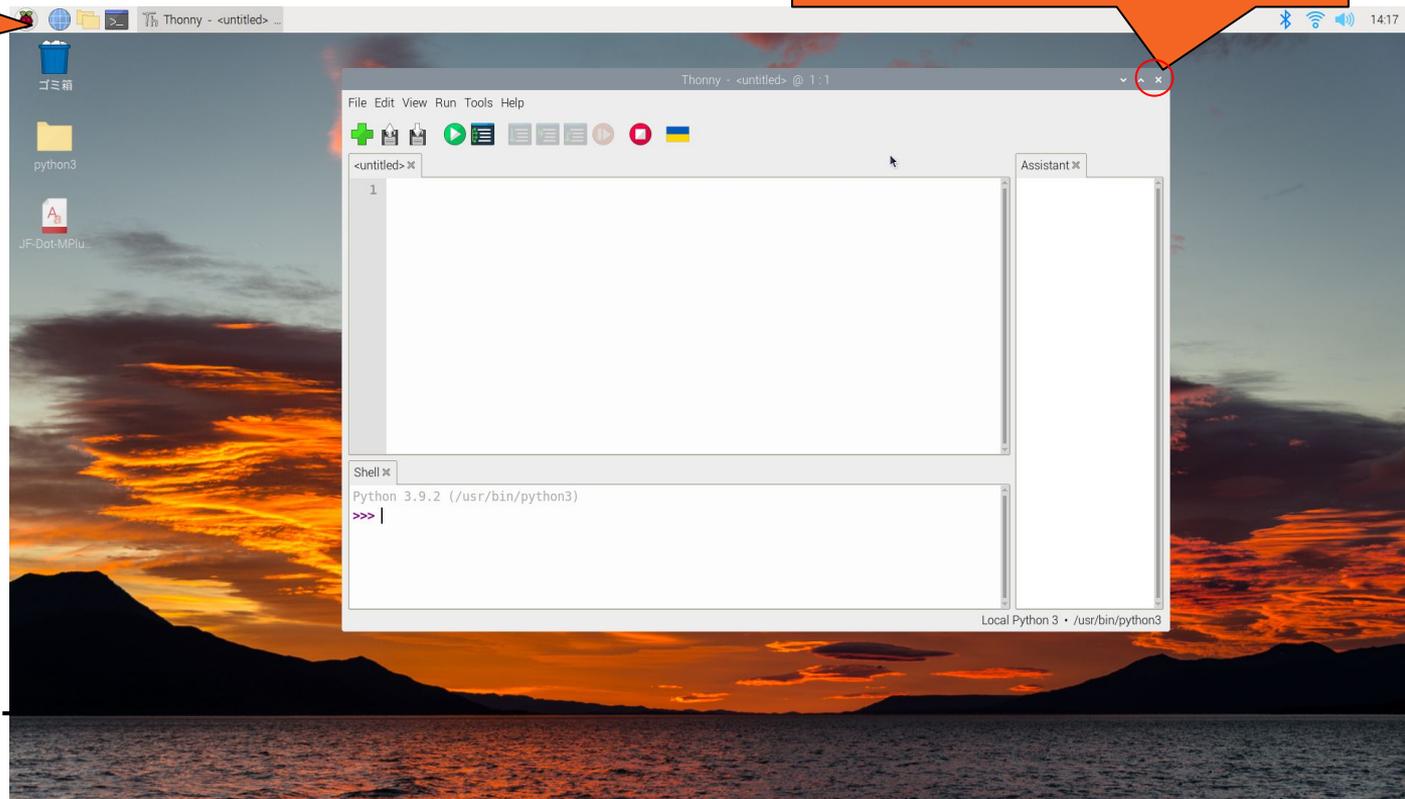


Thonnyエディタの初期設定



④6ページのように
再度開く

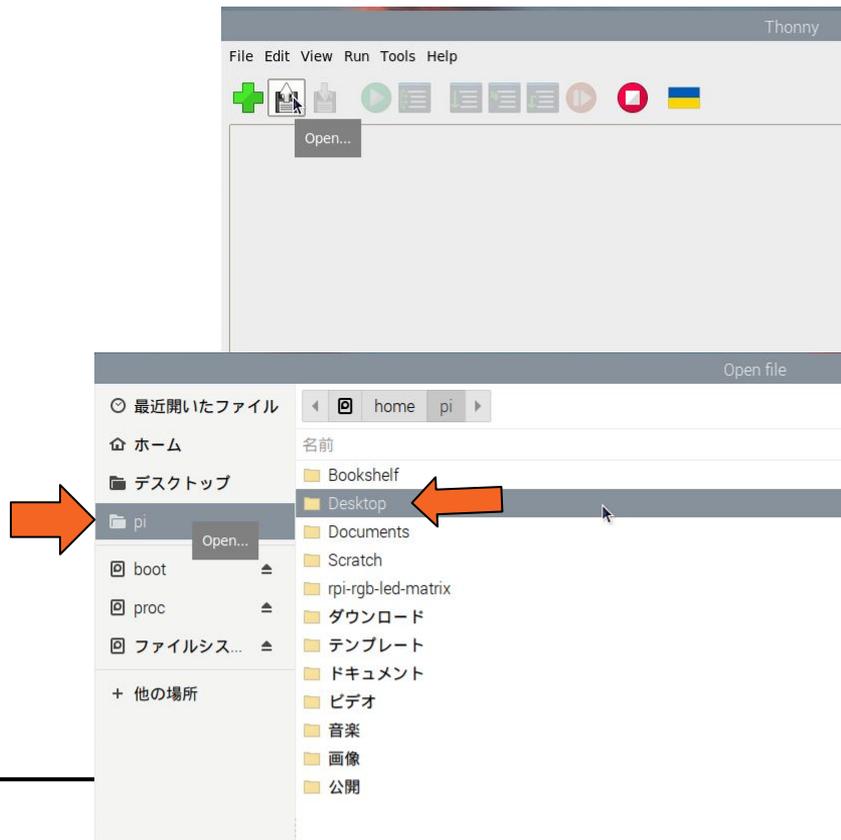
③バツを押して閉じる



サンプルコードの開き方

1 thonny上部の
このボタンをクリック

2 左の欄のpiを選択し
Desktopをダブルクリック

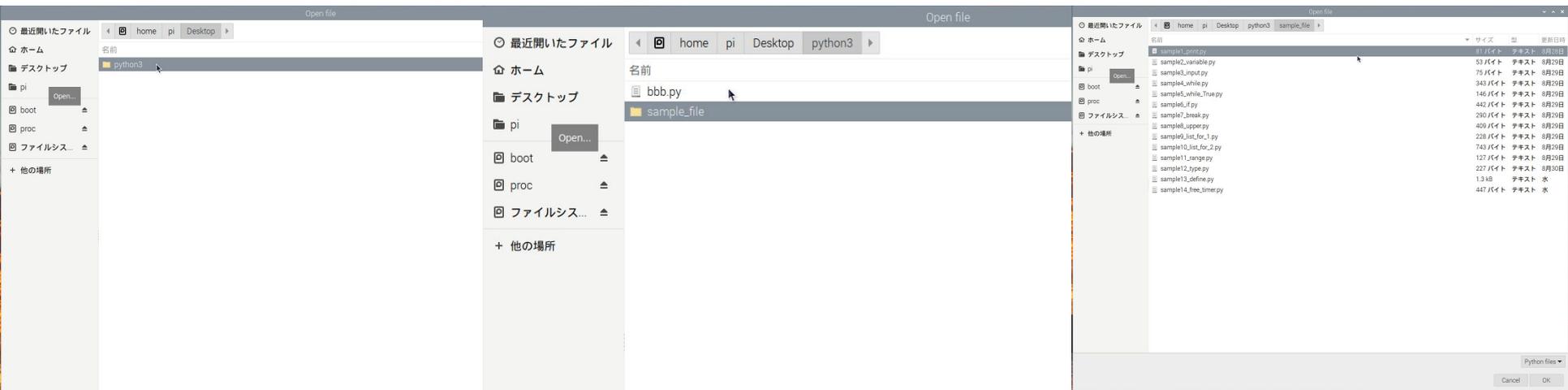


3 そのままpython3→sample_fileと開いて行くと

サンプルファイルの一覧があるので

指定されたものを開いていく

とりあえずsample1_printを開く



こんな感じのやつ



printについて

`sample1_print`を前ページで開いてもらいました

ここで緑の矢印ボタンを押してプログラムを動作させる

これは`print("この部分に書かれたことを下に出力する ")`

というものです

`print("")`この部分さえ変えていなければ

中身は自由に変更しても良いです

変数について

`sample2_variable`を開く

`print(test)`と内容は`test`となっていますが

実際に出力されるのは先ほどと変わらず

Hello Worldのままです

今回は`print`の中身が(“Hello world”)でした

今回は`(test)`だけです

でも出力結果は同じになっています

これはtestがただの「文字」ではなく「変数」だからです
変数は入れ物のようなものであり

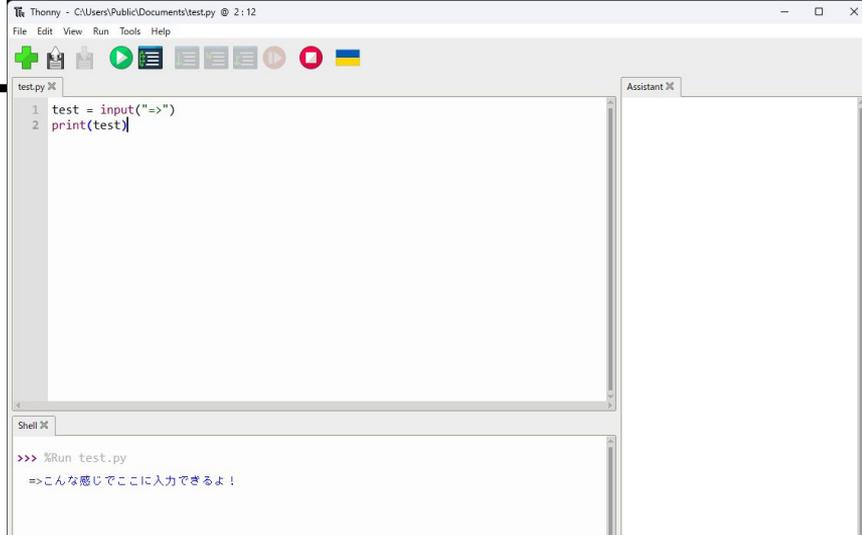
今回は”Hello world”を入れておいて
print(test)を動かす際に

testという変数の中身を読み込んで使いました

「”」これで、はさまれた文字はプログラムには影響せず
コンピューターがそのまま表示する文字として扱われます
なので、1と”1”という風に「”」の有無だけで
思った通りの動作がしなかったりします

次はinputを使ってみる

sample3_inputを開く



The screenshot shows the Thonny Python IDE interface. The main editor window displays a Python script named 'test.py' with the following code:

```
1 test = input("=>")
2 print(test)
```

Below the editor is a Shell window showing the execution of the script:

```
>>> %Run test.py
=>こんな感じでここに入力できるよ！
```

The Shell window also shows a prompt 'Assistant' on the right side.

これを実行すると入力ができるようになって

入力してEnterを押すと

入力した文字が表示されるはずです

inputは
test = input("=>")と書いた場合は、

testという変数にinputした内容を入れる

というものです

("=>")の部分は入力待機中のどこに入力するのか、
目印の表示の設定みたいなものです

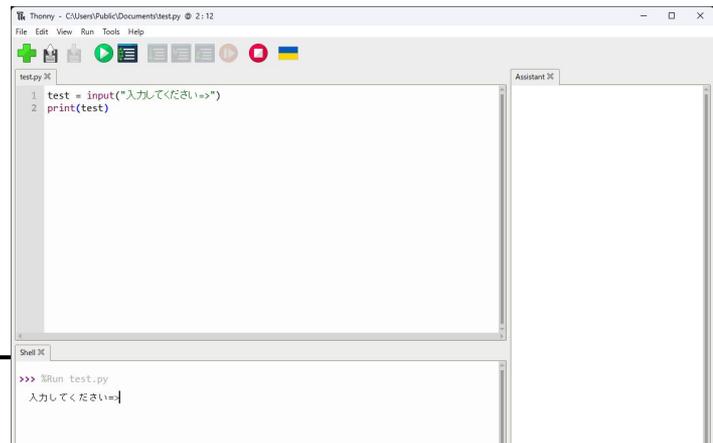
動作には影響しないので

極端な話空白でも動作はします

普通に文字とかも入れられます

基本的には

「=>」「:」とかが多いです



```
Thorny - C:\Users\Public\Documents\testpy @ 2:12
File Edit View Run Tools Help
testpy.py
1 test = input("入力してください=>")
2 print(test)

Assistant

Shell
>>> 実行 test.py
入力してください=>
```

inputについての大事な点

inputはその入力内容によって

動作したりするようにする前提だからなのか

input内容を変数に入れて利用します

一応そのままprintしたりもできますが

使うことはないです

このままだと、

一回入力したらもう一回実行しないと再度入力できない...

→繰り返させたい

whileを使う

whileとは

whileの後ろに書かれた条件が満たされている間は繰り返す
というようなものです

例えば...

[sample4_while](#)を開く

```
Thonny - C:\Users\Public\Documents\test.py @ 2:1
File Edit View Run Tools Help
+ 📄 🖨️ 🔄 🎮 📄 📄 📄 📄 📄 📄 🇺🇲

test.py ✕
1 import time
2 |
3 count = 0
4
5 while count < 11:
6     print(count)
7     count += 1
8     time.sleep(1)

Assistant ✕
The code in test.py looks good.
If it is not working as it should, then consider using
some general debugging techniques.
Was it helpful or confusing?

Shell ✕
>>> %Run test.py
0
1
2
3
4
5
6
7
8
9
10
```

こんな感じで
0~10までの数字が
順番に一秒ごとに
表示される

The screenshot shows the Thonny Python IDE interface. The main editor window displays a Python script named `test.py` with the following code:

```
1 import time
2
3 count = 0
4
5 while count < 11:
6     print(count)
7     count += 1
8     time.sleep(1)
```

The Shell window at the bottom shows the execution output:

```
>>> %Run test.py
0
1
2
3
4
5
6
7
8
9
10
```

An Assistant window on the right provides feedback: "The code in `test.py` looks good. If it is not working as it should, then consider using some general debugging techniques. [Was it helpful or confusing?](#)"

1
変数の中身を「”」を付けずに
数字にすると
数値を操作できる

2
`count += 1`
これを行うと
`count`変数の中の数値を+1
する

3
whileでcountの数値が11未満
なら繰り返すようにしている

The screenshot shows the Thonny Python IDE interface. At the top, there is a menu bar with 'File', 'Edit', 'View', 'Run', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for file operations and execution. The main window is divided into three panes:

- test.py**: Contains the following Python code:

```
1 import time
2
3 count = 0
4
5 while count < 11:
6     print(count)
7     count += 1
8     time.sleep(1)
```
- Assistant**: Displays a message: "The code in test.py looks good. If it is not working as it should, then consider using some general debugging techniques. [Was it helpful or confusing?](#)"
- Shell**: Shows the command prompt output:

```
>>> %Run test.py
0
1
2
3
4
5
6
7
8
9
10
```

-
- 4
print(count)をして
countの進みを可視化
これはなくても
while count < 11の
動作には関係ない
 - 5
import timeは
timeというのを
使えるようにするための
プログラムで
使えるようにする準備
 - 6
time.sleep(1)で
毎回一秒待機してから
次の繰り返しに行く
-

while Trueについて

whileの条件を入れるはずのところに
Trueと入れると常に条件を満たしている判定になり
無限ループになります
これを利用することはかなり多いです

ただwhile本来の

条件が満たされている間は繰り返す

という使い方も頭の片隅にあったほうがいいです

sample5_whileTrueを開く

このサンプルコードのようにすると

何回でも入力して表示することができます

無限ループしていて、勝手に停止しないので

停止したい場合は

上の赤丸に白い四角のボタンを押してください

ifを使ってみる

[sample6_if](#)を開く

実行してみる

testの内容を1、2、その他に自由に変更して

どんな動作をするか理解する

if ~

これはもしifの後ろに続く条件に合うならというものです

elif~

これは、ifの後じゃないと使えません

ifの条件に合わなかった場合elifの条件を参照します

else

これもifの後じゃないと使えません

これはifやelifなどすべての条件が満たされなかった場合、

動作します

ifをどう活用するかという...

[sample7_break](#)を開く

これを実行

ここで覚えてほしいのは

```
elif text == "exit":
```

```
    break
```

のbreak

breakをすると、

今いる無限ループを即座に終了して次に行くことができます

注意してほしいのは

プログラムを終了するものではない

ということです

inputに関して

大文字小文字などを間違えてしまうと動作しません

例えば頭だけ大文字にする人がいたりすると

それだけで動作しなくなります

→なのでinputで入力された文字をすべて大文字に変換して
大文字前提で動作するようにします

sample8_upperを開く

```
input("=>").upper()
```

この.upper()が大事

これを入れるとinputで入力された文字を

すべて大文字に変更して変数に入力する

ただ、これだとプリントする文字まで

勝手に大文字にしてしまう

次は list と for について

[sample9_list_for_1](#)を開く

listの内容を

forでlistの内容がなくなるまで繰り返し、

ほかの変数に入力してくれます

[sample10_list_for_2](#)を開く

使い方としては
好きな文字を好きなだけ入力した後、
exitと入力すると今まで入力したものを、
一覧で表示してくれる

```
Thonny - C:\Users\Public\Documents\test.py @ 13:9
File Edit View Run Tools Help
+ [Icons]
bbb.py test.py
1 test = []
2
3 while True:
4     inputstr = input("Enter your like food (or [exit] to quit)=>")
5     work = inputstr.upper()
6     if work == "EXIT":
7         break
8     else:
9         test.append(inputstr)
10
11 for f in test:
12     print(f)
13

Shell
>>> %Run test.py
Enter your like food (or [exit] to quit)=>餃子
Enter your like food (or [exit] to quit)=>麻婆豆腐
Enter your like food (or [exit] to quit)=>ラーメン
Enter your like food (or [exit] to quit)=>グラタン
Enter your like food (or [exit] to quit)=>小籠包
Enter your like food (or [exit] to quit)=>シューマイ
Enter your like food (or [exit] to quit)=>exit
餃子
麻婆豆腐
ラーメン
グラタン
小籠包
シューマイ
>>>
```

1

```
test = []
```

これでtestという枠を作る

こうすると上書きされず、

複数入れて置けるようになる

2

inputstr変数にinputで入力するが、

この段階で大文字に変換してしまうと

入力した文章も

大文字になっちゃうので注意

次はrange

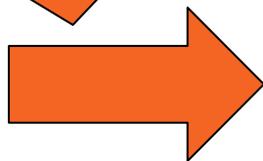
sample11_rangeを開く

forと組み合わせて～回だけ繰り返すというように

使えたりします

```
1 #これをやらないとこの場合はtimeが
2 #importは最初にまとめて書く
3 import time
4 #count変数の初期値を設定
5 count = 0
6 #count変数が11未満なら繰り返す
7 while count < 11:
8     print(count)
9     #count変数に+1される
10    count += 1
11    #一秒待機
12    time.sleep(1)
13
14 print("END")
```

count変数とかなしに
シンプルに~回繰り返すを実
現できる



```
2 import time
3 #rangeは0~()の中の数字-1まで出力する
4 for i in range(11):
5     print(i)
6     time.sleep(1)
7
8 print("END")
```

次は変数についての注意点について

sample12_typeを開く

変数に入力する際inputで入力したものは

string形式になっている

しかし四則演算等、数値として利用したい場合は

integer形式でないといけない

なので、string形式の内容をint()でinteger形式に

変更して利用している

関数について

すごく極端な話使わなくてもいいですが

違うところで繰り返し利用する操作を

毎度書くのも無駄なので、それをまとめることができます

プログラムの見やすさや、

理解しやすさのために利用されたりもします

ただ、これはあくまでプログラムが長くなってきたら

利用するものです

関数を増やしすぎるとプログラムが複雑になりすぎて

自分はもちろん、ほかの人もプログラムを理解しにくくなってしまうので

ほどほどにしましょう

sample13_defineを開く

これはinputした数字に応じて
タイマーを起動するプログラムです

def ~と頭にdefをつけると関数になります
そうするとdef~の~の部分を入れるだけで
その関数を呼び出すことができます

def ~ ()の()については次に説明します

`sample14_free_timer`を開く

これは関数の引数のサンプルコードです

inputした数字に応じて

その数字分の秒数のタイマーをセットする

プログラムです

`setting_time`変数に何秒タイマーにするかを入力し

`free_timer(setting_time)`と書くことで

`free_timer`関数を呼び出したとき

`setting_time`の情報を渡すことができます
