はじめてのプログラミング
～Python 編～

C タイプ

組込みコンピュータ Raspberry Pi 活用型

1　はじめに

プログラムとは？

　プログラムは、コンピュータにさせる処理を順番に書き出したものです。パソコン用のアプリ(デスクトップ・アプリケーション)やスマホ用のアプリ(モバイル・アプリケーション)、工場の機械の制御アプリ、ゲームなど、いずれも開発するためにプログラムが使われています。

　実際のプログラムは、英語と数字の組み合わさった不思議な文字列が並んだものとなっていますので、はじめはとっつきにくく感じるかもしれません。しかし、この教材では、ちょっとずつ段階的に実習を通して学んでいきますので、徐々に慣れていくでしょう。きっと、最後のページまで進んだときには、この教材の学習目標への到達を実感していると思います。

2　学習目標

　この教材の学習目標は、次の通りです。

　値や変数、配列(リスト)を用いた順次処理や条件分岐、繰り返し処理を組み合わせたPython のプログラムを作成できる。

1. Python のプログラムを入力し、動作確認できる。
2. 順次処理のプログラムを作成できる。
3. 変数について説明できる。
4. 変数を使ったプログラムを作成できる。
5. 条件分岐のあるプログラムを作成できる。
6. 繰り返し処理のプログラムを作成できる。
7. 配列(リスト)を使ったプログラムを作成できる。

豆知識　Python って何？

　Python は、シンプルなプログラム記述ができる無料のプログラム言語です。様々な機能拡張をもたらすライブラリが充実していますので、機械学習、web アプリにも使われています。Windows パソコン、mac、組込みコンピュータの Raspberry Pi でも同じように使えます。Python の人気は広がり、近年ではソフトウェア開発者や NASA、CERN、Google など世界中の多くのプログラマーに使われるようになっています。ちなみに Python 開発者のグイド・ヴァンロッサム氏は、BBC のコメディ番組「空飛ぶモンティ・パイソン」のファンであったこともあり、いたずら心から言語の名前を Python としたそうです。ということからすると、プログラム言語 Python は、英語の意味であるニシキヘビとは関係ないようです。

(参考)ウィキペディア(グイド・ヴァンロッサム), https://ja.wikipedia.org/

3 プログラム作成と実行

プログラムをどうやって作成し、実行するの？

Python というプログラム言語とプログラム作成のためのアプリケーションは、すでにインストール済みです。このテキスト「はじめてのプログラミング～Python 編」では、Python のプログラムを作成に便利な無料のエディターVisual Studio Code を使って、プログラミングの学習を進めていきます。

なお、プログラミング手順は、大きく6つの段階で進めていくことになります。

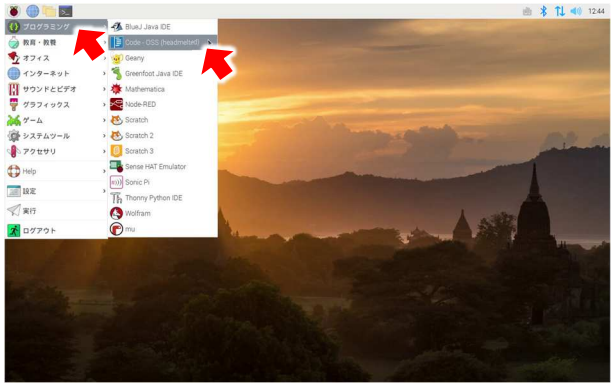
それでは、コンピュータの電源を入れて、プログラミングを始めましょう。

まず、エディター「Visual Studio Code」を使った Python のプログラム作成と動作確認の流れをビデオでつかんでいただきたいと思います。次の QR コードにインターネット上のビデオへのリンクを載せています。スマホをかざして、ビデオを見てください。

RasPi01

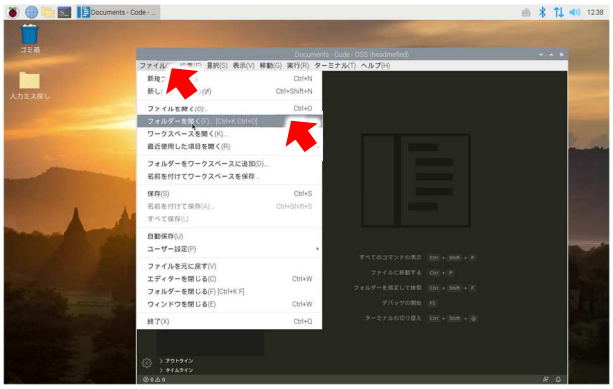
①エディターの起動

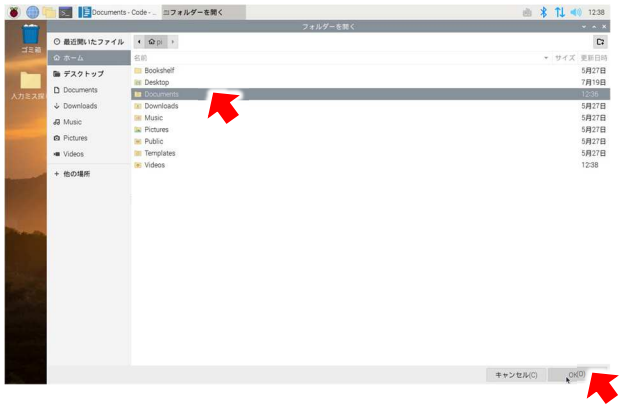
プログラム開発環境として活用するエディター「Visual Studio Code」を起動するには、スタートメニューからプログラミング ＞ Code-OSS をクリックします。



②新規プログラム・ファイルの用意

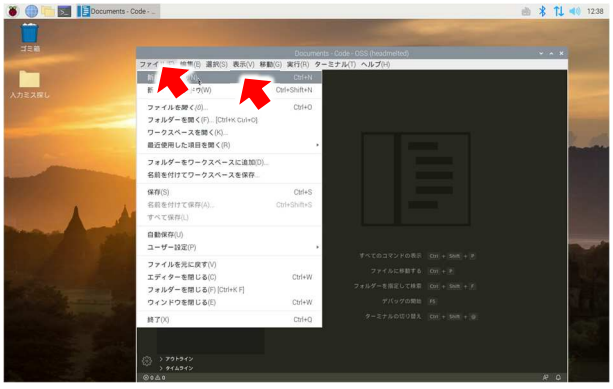
ファイル ＞ フォルダーを開く をクリックし、 Documents をクリック、[OK]ボタンをクリックします。

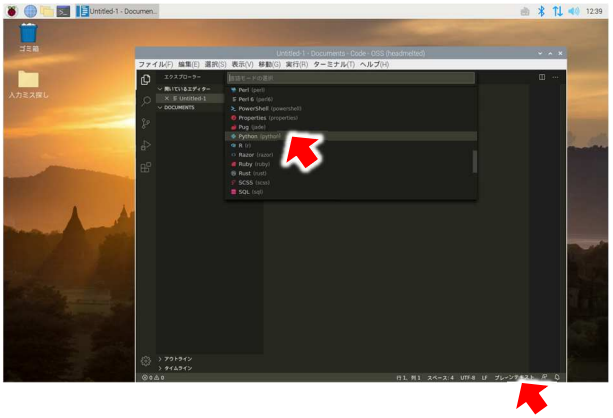




ファイル ＞ 新規ファイル をクリックします。

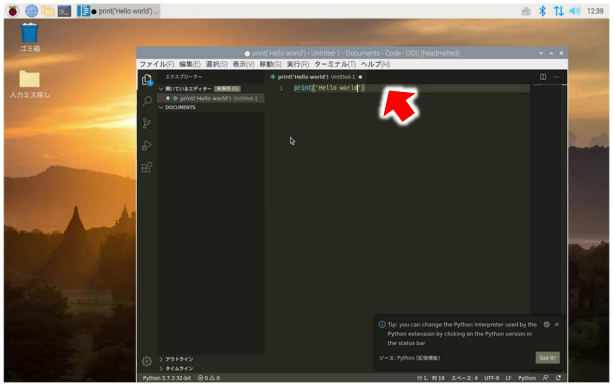
右下のプレーンテキストと書かれた場所でクリックし、「Python」をクリックします。





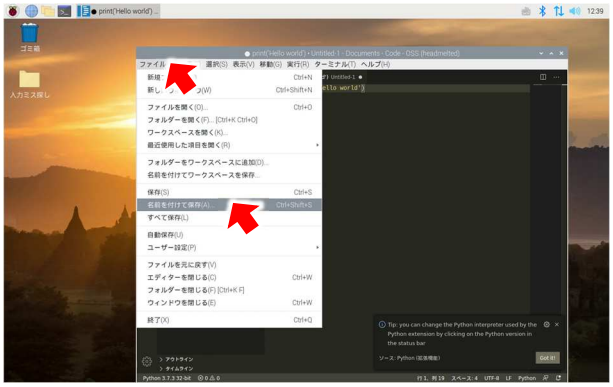
③プログラムの記述

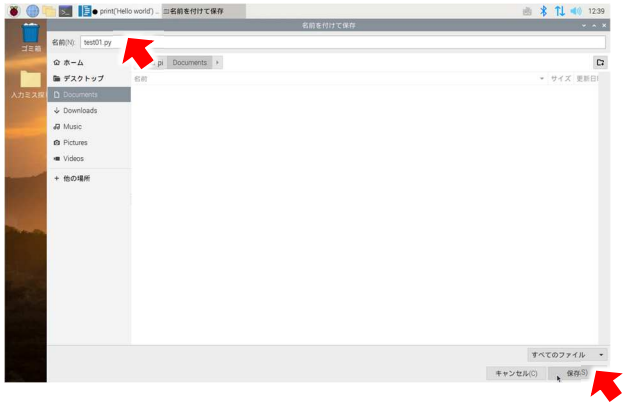
プログラムを記述します。



④プログラム・ファイルの保存

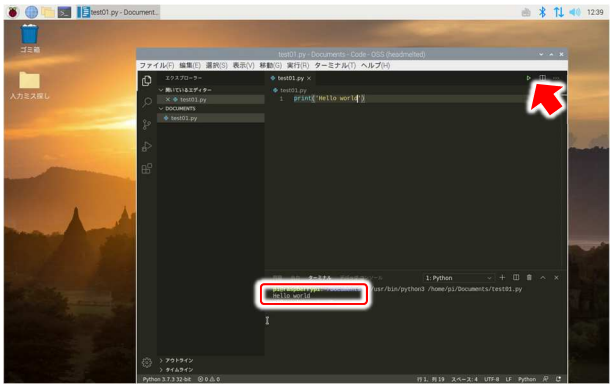
ファイル ＞ 名前を付けて保存 をクリックし、test01.py など最後に".py"が付いたファイル名にして[保存]ボタンをクリックします。





⑤プログラムの実行

右上にある▶ボタンをクリックします。(実行結果は画面右下に出ます。)



⑥エディターの終了

エディター「Visual Studio Code」の終了をするには、「Visual Studio Code」のウィンドウの右上にある[×]ボタンをクリックします。

では、エディターを使って、次に示す例題 01 のプログラムを作成してみましょう。

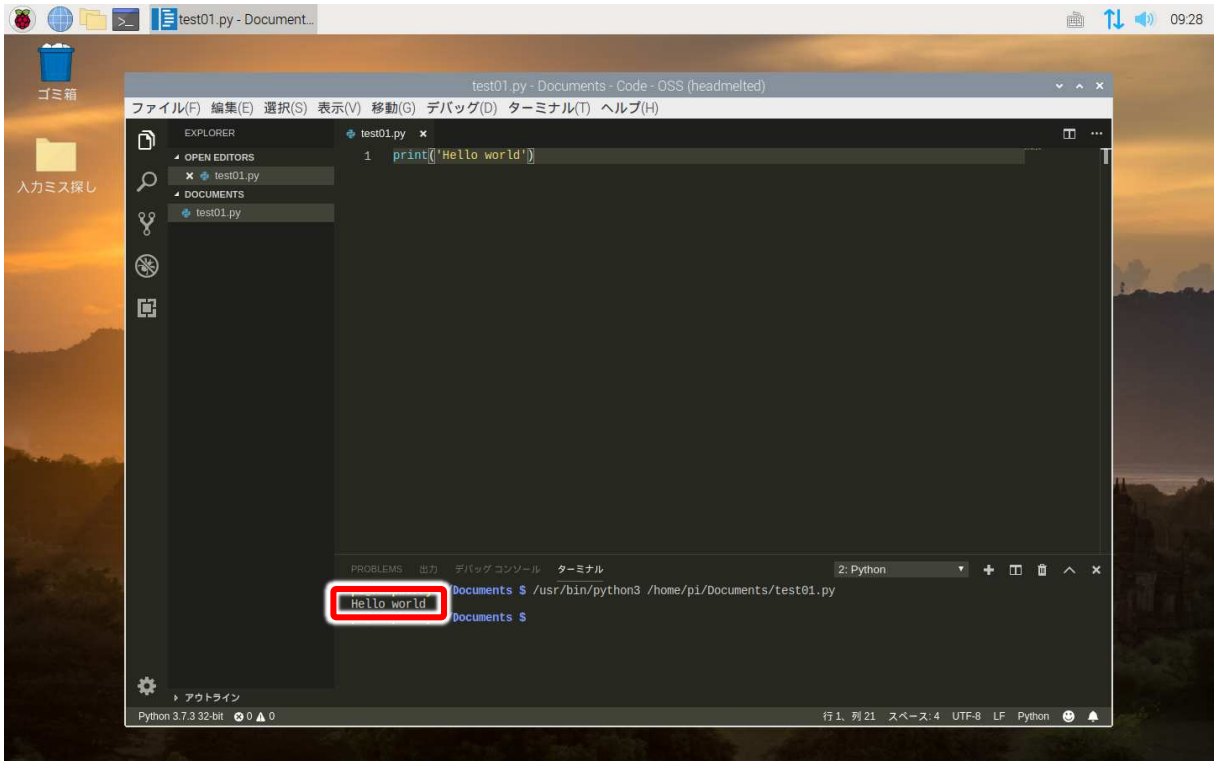
P.3 に示した①エディターの起動　から P.4 に示した③プログラムの記述の操作をします。

ここでは黒い太枠で囲まれた部分をキーボードで入力します。
黒い太枠の横に示している「１」は１行目を示しています。プログラムではありませんので、キーボードで入力する必要はありません。

例題 01　test01.py

```
1  print('Hello world')
```

プログラムの入力が終わったら、“test01.py”としてプログラム・ファイルを保存し、実行してください。
実行すると、次の図の位置に、実行結果が表示されるはずです。



実行結果

```
Hello world
```

例題 01 は、print('Hello world')の 1 行だけではあるものの、Hello world を画面に表示をするプログラムです。

プログラムで使った print ( ) というのは、( ) 内の内容を表示させる Python の命令です。print ( ) 関数と呼びます。

関数は、

関数名(引数(ひきすう))

の形になっています。

例題 01 では、関数名は print で、引数は'Hello world'です。
関数は、引数が渡されて何らかの働きをするものになっています。

print()関数

引数として渡した値を表示する関数です。

使い方
print(文字列)

引数に文字列を使う場合、文字列をシングル・クォーテーションで囲む(またはダブル・クォーテーションで囲む)ことがルールとなっています。

(使用例)

```
print('Hello world')
print("Hello world")
```

つまり、print ( ) の使い方を踏まえると、print ( ) 内の内容を変えれば、表示内容が変わるということになります。

既にある test01.py をちょっと直して、test02.py としてプログラミングしてみましょう。

操作の流れは、スマホで次の QR コードに載せたビデオから確認してください。

RasPi02

プログラムの実行をするとファイルを上書き保存してしまうので、実行する前にファイルを保存するようにしてください。

例題 02　test02.py

```
1  print('Hello world') ⏎
2  print('Computer Programming for Everybody')
```

※　⏎は、「[Enter]キーを押して、改行してください」という印です。

実行結果

```
Hello world
Computer Programming for Everybody
```

2 行表示されました。

プログラムは、コンピュータにさせる処理を順番に書き出したものです。

上から下へ命令を書くことで、コンピュータは順番に処理してくれるわけです。

Python では、基本的に 1 行に 1 つの命令を書きます。

今度は、文字列に日本語を使ってみます。
test02.py を直して、test03.py としてプログラミングしてみましょう。

例題 03　test03.py

```
1  print('おはよう') ⏎
2  print('こんにちは') ⏎
3  print('こんばんは',␣'おつかれさまです')
```

日本語を入力する部分だけ、日本語入力をオンにしてプログラムの記述をしてください。
なぜなら、print という文字だけでなく()、'や,、スペース(空白)でさえも日本語入力オンのまま記述してしまうと全角文字になり、構文エラーになってしまうからです。
(構文エラーは次ページで解説します)

※ 3 行目の ␣ は、(日本語入力がオンでないときの文字(半角))スペースの印です。このテキストでは目立つように ␣ で示すことにしました。スペースを押してもこの文字は出ません。

※ プログラムのなかでカンマ「,」を使うときには、読みやすくするため、カンマ「,」の後に半角スペースを 1 つ入れることをお勧めします。(詳しくは、P.22 Python コーディング規約 【PEP8】で説明します。)
なお、カンマ「,」の後の半角スペースをなくしても、増やしても構文エラーにはなりません。

実行結果

```
おはよう
こんにちは
こんばんは おつかれさまです
```

print()関数に渡す引数は、カンマ(,)で区切れば複数にすることができます。引数の内容は、スペースで区切られ表示されます。

print()関数

(使用例)

```
print('Hello', 'world')
print('年月日', '最高気温', '最低気温', '天気')
```

(実行結果)

```
Hello world
年月日 最高気温 最低気温 天気
```

エラーについて

プログラミングをしていると、入力ミスや関数などの使い方のミスなどからエラーが起こることがあります。もしエラーを起こしたなら、プログラムからミスを見つけ、修正することが必要になります。

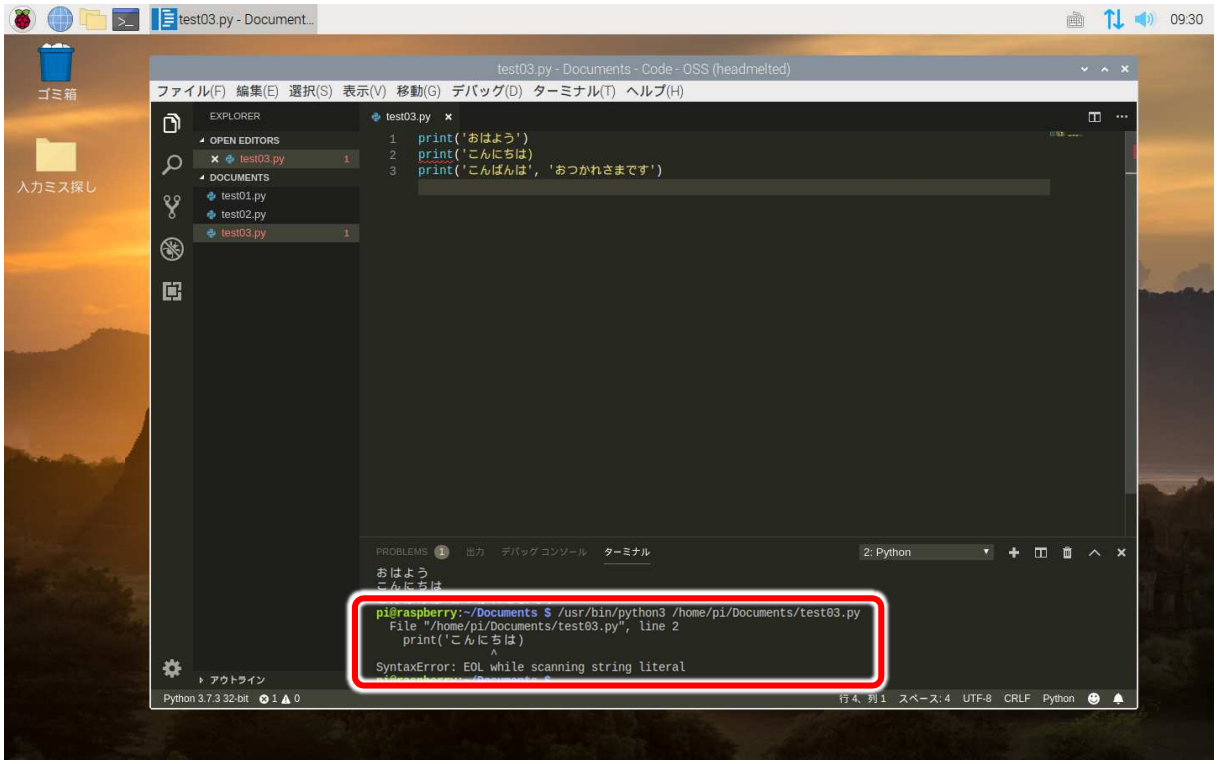
ここで、エラーをわざと起こし、対処する流れを体験しましょう。
どこに、どのようなエラー表示がされるのか注目してください。

例題 03 をもとに作業します。
2行目の「print('こんにちは')」を赤い文字部分「print('こんにちは)」のようにします。つまりシングル・クォーテーション'を削除するだけです。

例題 03　test03.py

```
1  print('おはよう') ⏎
2  print('こんにちは) ⏎
3  print('こんばんは',␣'おつかれさまです')
```

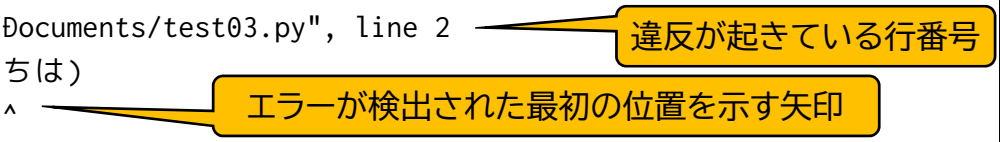
実行してください。
次の図の赤い枠の部分に英語でエラー・メッセージが表示されます。



赤い枠の部分には、次のようなことが書かれています。

（エラー例）

```
pi@raspberry:~Documents $ /usr/bin/python3 /home/pi/Documents/test03.py
  File "/home/pi/Documents/test03.py", line 2
    print('こんにちは)
                  ^
SyntaxError: EOL while scanning string literal
```



エラー・メッセージには、違反の起きている行番号やエラーが検出された最初の位置を示す小さな「矢印^」があります。プログラムの書き方のミスが、違反の起きている行番号の「矢印^」の場所か、その周辺にあることを意味します。

なお、起きているエラーは、SyntaxError という構文エラーです。構文エラーとは、プログラムの記述が Python のルールにあっていないことによるエラーです。

もっと具体的に言えば、SyntaxError とは

SyntaxError
余計なカッコ()やクォーテーション”” ’’などがある、あるいは足りないことなどによる構文エラー
（対処方法） 矢印周辺にあるルールにあっていない命令文をみつけ、修正します。 ● 余計なカッコやクォーテーション、あるいは囲み忘れは、ありませんか？

文字列はシングル・クォーテーションで囲む（またはダブル・クォーテーションで囲む）ことがルールとなっているのに、削除したため囲まれていないからエラーになったわけです。

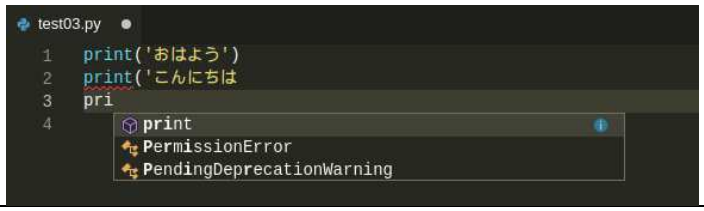
削除したシングル・クォーテーション'を追加し、２行目が「print('こんにちは')」となるように修正してください。

※ 別冊「エラーと対処方法」には、主なエラーと対処方法が載っています。
これからエラーに出会った場合、別冊「エラーと対処方法」を見て、対処してください。

豆知識　Visual Studio Code の色分け機能と入力補完機能

エディターVisual Studio Code は、Python の構文に沿った色分けをしてくれます。それだけではなく、構文エラーと思われる箇所に赤い波線を付けてもくれます。ミスに気づきやすくしてくれる機能です。

また、入力を補完してくれる機能もあります。途中までキーボードで打つと候補が出てくるときがあるのではないでしょうか。入力したい文字列と合っていれば[tab]キーを押します。すると文字列が出てきます。これもミスを減らしてくれる機能です。



それでは、これまで学んだ内容をもとに次の課題にチャレンジしてください。

チャレンジ　test04.py

次のメッセージを表示するプログラムを作成してください。

（実行結果）

```
実行させたい命令を
上から下へと
順番に記述します。

プログラムの記述は、基本的に日本語入力をオフにしておこないます。
```

いかがですか？

※ もし答えが知りたい場合は、デスクトップにある「入力ミス探し」フォルダーから、プログラム名が含まれるリンク（ここでしたら　test04 の入力ミス探し.html　）を見つけ、プログラムから開く　＞　Google Chrome　をクリックしてください。

1 行ずつプログラムを記述していくことに慣れたのではないかと思いますので、以降の例題では、プログラムを示す黒い太枠 と ⏎の表記を省略します。

4 さまざまな計算

計算するプログラムを作成してみましょう。電卓よりも便利な使い方ができます。
次に示す例題 05 のプログラムを作成してください。

例題 05　test05.py

```
1  print('計算')
2  print(60␣*␣60␣*␣24)
3  print((100␣+␣100)␣*␣1.08)
```

※ 2 行目の 60␣*␣60␣*␣24 や 3 行目の(100␣+␣100)␣*␣1.08 の式に見るように、「+-*/=」など計算に用いる演算子の前後には読みやすくするため半角スペースを 1 つ入れることをお勧めします。(詳しくは、P.22 Python コーディング規約 【PEP8】で説明します。)
なお、演算子前後の半角スペースはなくしても、増やしても構文エラーにはなりません。

実行結果

```
86400
216.0
```

print()関数の引数には、文字列だけでなく、値や計算式を渡すこともできます。

print()関数

引数として渡した値を表示する関数です。

使い方

```
print(値)
print(計算式)
```

引数に数値や数式を使う場合は、クォーテーションはいりません。
丸カッコを使った計算式では、数学と同じように丸カッコ内が他の計算より優先されます。ただし、丸カッコの囲みができていないと不適切な引数となり、エラーになりますので、注意してください。

(使用例)

```
print(100)
print(100 * 100)
```

ちなみに、Python での計算に使える主な演算子は次の通りです。

Python の主な演算子（x と y の演算の場合です、赤い文字に注目してください）

演算子	意味	使用例	使用例の結果
x + y	x と y の足し算	10 + 5	15
x - y	x と y の引き算	10 - 5	-5
x * y	x と y の掛け算	10 * 5	50
x / y	x を y で割った商	10 / 4	2.5
x % y	x を y で割った余り	10 % 4	2
x // y	x を y で割った商の 小数点を切り捨てたもの	10 // 4	2
x ** y	x の y 乗	4 ** 2	16
- y	x の符号反転	- 5	-5

それでは、これまで学んだ内容をもとに次の課題にチャレンジしてください。

チャレンジ　test06.py

1 行目に次のメッセージを表示し、

```
合計金額
```

2 行目に次の計算式の結果を表示するプログラムを作成してください。

$$(50 + 200 \times 3) \times 1.08 + 390 \times 1.1$$

※　50 円の卵と 200 円の牛乳 3 本には 8%の消費税、390 円のお酒には 10%の消費税。その合計は？

ここまでの学習の振り返り

ここまでの学習におけるポイントを確認しましょう。

- プログラムは、(コンピュータへの)命令の集まり。
- コンピュータはプログラムに記述された命令を上から下へと順番に処理していくので、プログラムには、コンピュータにさせたい命令を上から順に記述する。

確認クイズ1

次の QR コードには、ここまで学んだプログラムに関するクイズを載せています。スマホで QR コードを読み、確認クイズにより自身のプログラミングに関する理解度をたしかめてください。

次の目標に到達できましたか？

- Python のプログラムを入力し、動作確認できる。
- 順次処理のプログラムを作成できる。

5　値と変数、型

　プログラムのなかで、整数・小数点・文字列など値を格納できる入れ物があります。変数と呼ばれます。

　a と b の 2 つの変数を用いたプログラムの作成と確認を通して、使い方を確かめてみましょう。

例題 07　test07.py

```
1  a␣=␣500
2  b␣=␣28
3  print(a␣*␣b)
```

実行結果

```
14000
```

変数の作り方

　Python では、さまざまな変数を作ることができます。ただし、変数名のつけ方にはルールがあります。

変数

　Python では、変数名のつけ方に構文上のルールがあります。

- 変数に使うことができる文字の種類は、アルファベット、数字、アンダースコア(_)だけ
- 変数名のアルファベットの大文字と小文字は区別される(abc と ABC は異なる変数)
- 変数名の先頭に数字を使うことはできない
  (つまり、A1 という変数は使えるが、1A はエラー)
- Python のプログラムに用いられる特別な単語(予約語といいます)を変数として使うことはできない

特別な単語(予約語)は、次の通りです。

特別な単語(予約語)						
False	None	True	and	as	assert	break
class	continue	def	del	elif	else	except
finally	for	from	global	if	import	in
is	lambda	nonlocal	not	or	pass	raise
return	try	while	with	yield		

それでは、これまで学んだ内容をもとに次の課題にチャレンジしてください。

チャレンジ　test08.py

次のプログラムを入力して実行したところ、エラーが出ました。
エラーなく動作(13860 と表示)するよう修正をしてください。

```
1  muscat␣=␣4980
2  peach␣=␣780
3  print(Muscat␣*␣2␣+␣Peach␣*␣5)
```

※ もしエラーの原因が見つからない場合は、別冊「エラーと対処法」 の 「2 プログラム・ミスの箇所が見つからない場合の対処方法」を参照ください。確認方法を示しています。

変数に小数点を格納することもできます。

例題 09　test09.py

```
1  wheel_inch␣=␣24.0
2  wheel_cm␣=␣wheel_inch␣*␣2.54
3  print(wheel_inch,␣'インチは、',␣wheel_cm,␣'cm です')
```

※ プログラム中の wheel_inch などの_(アンダースコア)は、[Shift]キーを押しながら、[_\ろ]キーを押します。

実行結果

```
24.0 インチは、 60.96 cm です
```

変数に文字列を格納することもできます。

例題 10　test10.py

```
1 muscat = 'マスカット'
2 peach = 'もも'
3 print(muscat)
4 print(muscat + peach)
5 print(muscat * 2 + peach * 2)
```

実行結果

```
マスカット
マスカットもも
マスカットマスカットももも
```

変数に文字列を格納することもできますが、実行結果のように、文字列同士の場合、足し算は、文字列の接合、掛け算は、掛けた数分の文字列なってしまいます。

構文エラーではありません。数値と文字列によって処理が異なるために、このような結果になるのです。

コンピュータが愚直にプログラム内容を実行した結果なのですが、値の種類によって処理が異なることを知らないと、おかしな結果を出た感じることでしょう。

次に示す例題 11 のプログラムを作成してください。

例題 11　test11.py

```
bento_price␣=␣'500'
student_number␣=␣28
print(bento_price␣*␣student_number)
```

実行結果

```
50050050050050050050050050050050050050050050050050050050050050050050050050050
0500500500500500
```

やっぱり起きました。

bento_price は、'500'という文字列として格納されていますから、28 回繰り返し表示されたというわけです。

500*28 のような数値の掛け算にするには、どのようにしたらいいのでしょうか？

実は、プログラムで用いる変数やデータ(値)は、型（かた）をもっています。

これまでに登場したデータの型は、次の3つです。

(組込みデータ)型	データの書き方	例
整数型　int 型	小数点のない数値	-1　0　123
小数型　float 型	小数点を付けた数値	3.14159　0.08　10.0
文字列型　str 型	シングル・クォーテーションまたはダブル・クォーテーションで囲んだ値	″マスカット″　″もも″ “500″

これらの型を別の型に変更することで、対処しましょう。

今回の場合は、文字列型を整数型に変更して対処します。

別の型への変更は、型変換をする関数で実現できます。

型変換をする関数　（変数 X を使った場合）

関数	意味	使用例	使用例の結果
int(X)	X を整数に変換	int('123')	123
float(X)	X を小数に変換	float(123)	123.0
str(X)	X を文字列に変換	str(123)	"123"

文字列型を整数型に変更するのは、int()関数です。引数に文字列を渡すと、整数を返します。

例題 11 の 1 行目を次のように変更して、実行結果を確認してください。

```
1  bento_price␣=␣int('500')
```

実行結果

```
14000
```

※ もっとも、int()関数にも限界はあります。引数として数値にならない値(たとえば'abc')を渡すと ValueError というエラーになります。

さて、ここでキーボードからの入力を取り込む関数である input()関数を使ってみましょう。
引数に文字列を渡すとキーボードからの入力を待つときに表示されます。

input()関数

キーボードからの入力を取り込み、文字列として返す関数です。
引数には、キーボードからの入力を待つときに表示するための文字列を渡します。
※ 引数を省略することもできますが、何も表示されずキーボードからの入力待ちになるのでお勧めしません。

使い方

```
変数 = input(文字列)
```

（使用例）

```
nedan = input('お値段は？')
print(nedan)
cat_name = input('お名前は？')
print(cat_name)
```

（実行結果）

```
お値段は？29800
29800
お名前は？まだない
まだない
```

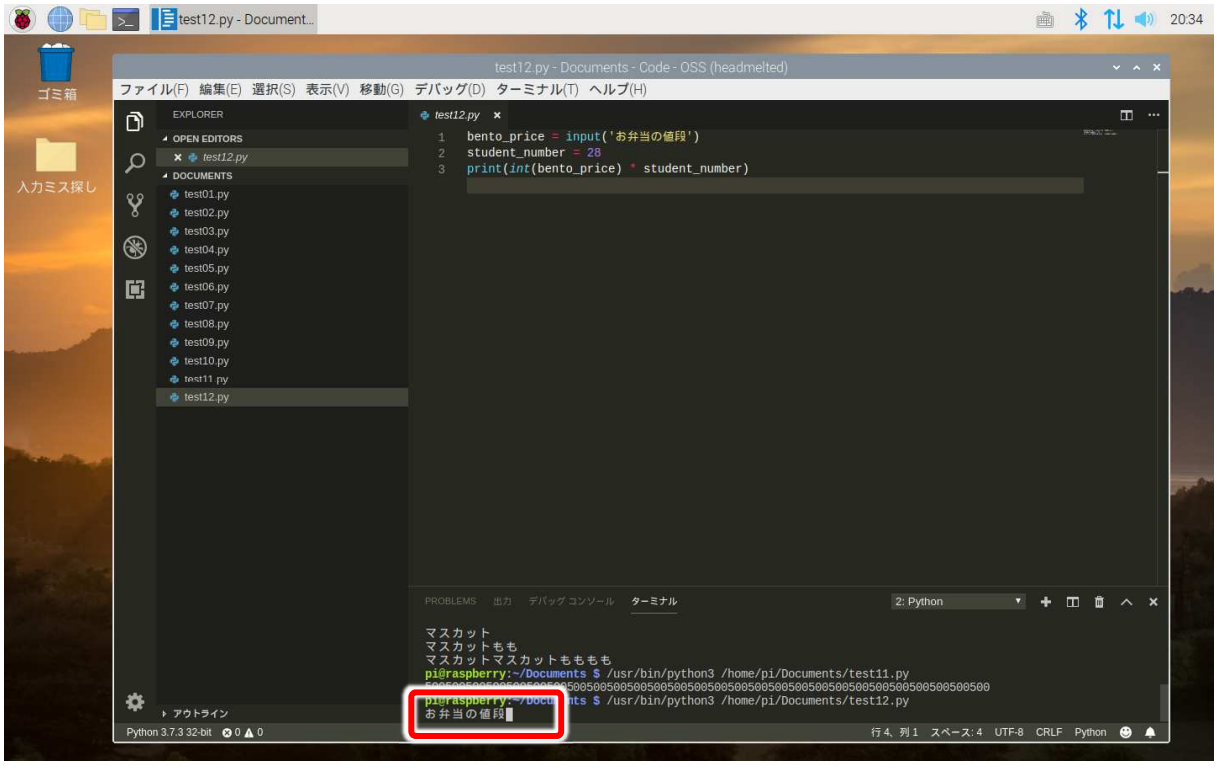
この input()関数には、返り値があります。使用例で示したように、input()関数はキーボードから取り込んだ文字列を返すので、変数には、その文字列が代入されることになるのです。

つまり、input()関数により変数へ文字列型の値が代入されることになるわけです。

さあ、それでは input()関数を使ったプログラムを作成してみましょう。
事前に全体の流れを、次のビデオで確認してください。

RasPi03

例題 12　test12py	
1	bento_price␣=␣input('お弁当の値段')
2	student_number␣=␣28
3	print(int(bento_price)␣*␣student_number)



実行結果　例(緑の文字は入力例)

```
お弁当の値段 500
14000
```

わかりやすいプログラムにする方法

だんだんプログラミングに慣れてきたと思います。エラーを避けるプログラムにすることが最も重要ですが、それだけでなく自分にも他人にもわかりやすいプログラムにすることも重要です。

わかりやすいプログラムにするためにどうすればよいのでしょうか。そのひとつに変数名に意味を持たせることがあります。プログラムが長くなったり、複数で作ったりすると変数名を間違えることが生じがちです。間違えのないプログラムにするためにも意味がとれる変数名にすることが重要です。

また、読みやすいプログラムにすることも大切です。Python では、読みやすいプログラムにするために字下げやスペースなどについてもルール化した「コーディング規約 PEP8【PEP8】」を定めています。このテキストでは、PEP8 を意識したプログラムにしています。ぜひ、Python プログラミング時に心がけてください！

豆知識　読みやすいプログラムのための Python コーディング規約【PEP8】

プログラムは、動作させるために使われるだけでなく、別のプログラム作成の参考にも使われもします。プログラムを読みやすくするために、プログラムの記述方法を示した Python コーディング規約 PEP8 が設けられています。主なものを示します。

- 変数名は、小文字のみで、単語間をアンダースコア(_)で区切る
- カンマ(,)の後にはスペースをつける
- 代入(=)や他の演算子(+-/*など)を揃えるために、演算子の周囲に１つ以上のスペースを入れる
- 字下げ(インデント)は半角スペース ４ つで統一する(タブではなくスペースを使用する)
- 文字列の引用符は、シングル・クォーテーションでもダブル・クォーテーションでもよい
- 変数が予約語などと変数名が被るかもしれないと思ったら、最後にアンダースコアを付ける

(参考)Python コーディング規約 PEP8, https://pep8-ja.readthedocs.io/ja/latest/

変数の値は更新される

次に示す例題 13 のプログラムを作成してください。

例題 13　test13py

```
1  bento_price␣=␣500
2  print('改訂前の値段:',␣bento_price)
3  bento_price␣=␣380
4  print('改訂後の値段:',␣bento_price)
```

実行結果

```
改訂前の値段: 500
改訂後の値段: 380
```

このプログラムで、変数の値が更新されることに気づいたでしょう。
同じ変数名でも、値の代入を繰り返すと、値はそのたびに更新されます。

変数への値の代入は、変数＝値　と記述します。

※ 変数＝値ではなく逆に、　値＝変数　とするとエラーになるので注意してください。
　たとえば、 500 = bento_price　とすると構文エラー(SyntaxError)になります。

変数の値が更新されることを踏まえて、次の課題にチャレンジしてください。

チャレンジ　test14.py

キーボード入力により、お弁当の値段を改定するプログラムを作成したいと思います。
次のプログラムのオレンジ色の空欄を埋めてください。

```
1  bento_price␣=␣500
2  print('改訂前の値段:',␣bento_price)
3  bento_price␣=␣
4  print('改訂後の値段:',␣bento_price)
```

ここまでの学習の振り返り

ここまでの学習におけるポイントを確認しましょう。

- 変数名の付け方にはルールがある
  - 変数名に使える文字の種類は、アルファベット、数字、アンダースコア(_)だけ
  - 変数名のアルファベットの大文字と小文字は区別される
  - 変数名の先頭に数字を使うことはできない
  - 特別な単語(予約語)を使うことができない
- 変数に値を代入するには、変数＝値　と書く
- 変数は更新される
- わかりやすいプログラムとなるよう「意味を反映した」変数名を用いる

確認クイズ2

次の QR コードには、ここまで学んだプログラムに関するクイズを載せています。スマホで QR コードを読み、確認クイズにより自身のプログラミングに関する理解度をたしかめてください。

次の目標に到達できましたか？

- 変数について説明できる。

6　条件によって処理を分ける方法(if 文)

プログラムでは、「もし、～という条件が成立するならば･･･させる」という処理ができます。このような処理は条件分岐と呼ばれ、Python では if を使います。使い方は、次の通りです。

if 文

もし、～という条件が成立するならば･･･させるという処理です。

使い方

```
if　条件式:
␣␣␣␣条件式が成立したときの処理
```

(使用例)

```
if x==1:
␣␣␣␣print('x は 1 です')
␣␣␣␣print('変数 x の値は 1 です')
```

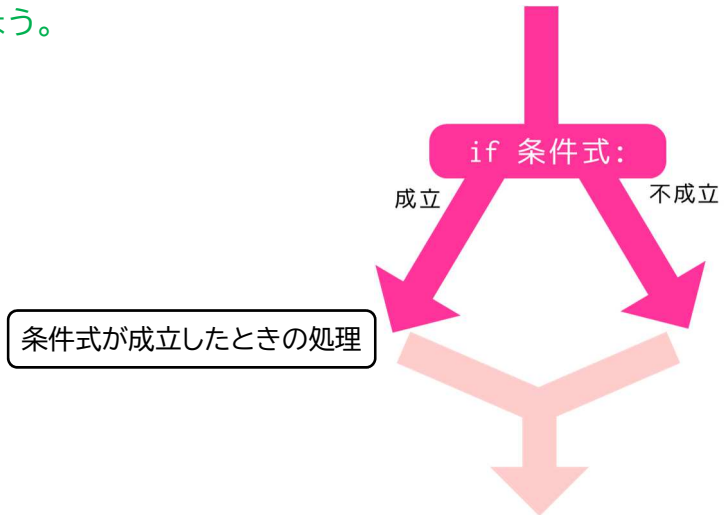
if 文の行末にコロン:を忘れずに記述するところと、条件式が成立したときの処理を、半角スペース 4 つ分の字下げ(インデント)するところがポイントです。

※　エディターVisual Studio Code では、if の行で改行すると自動的に半角スペース 4 つ分字下げされます。これは【PEP8】に準拠した半角スペース 4 つ分の字下げです。(P.22)

条件式は、次の比較演算子を用いて作ります。

比較演算子	意味	数学記号	条件式の例
x == y	x が y と等しいならば、	=	a == 365
x != y	x が y と異なるならば、	≠	a != 'Yes'
x < y	x が y より小さいならば、	<	a < 0
x <= y	x が y 以下ならば、	≦	a < =100
x > y	x が y より大きいならば、	>	a > 170
x >= y	x が y 以上ならば、	≧	a >= 60

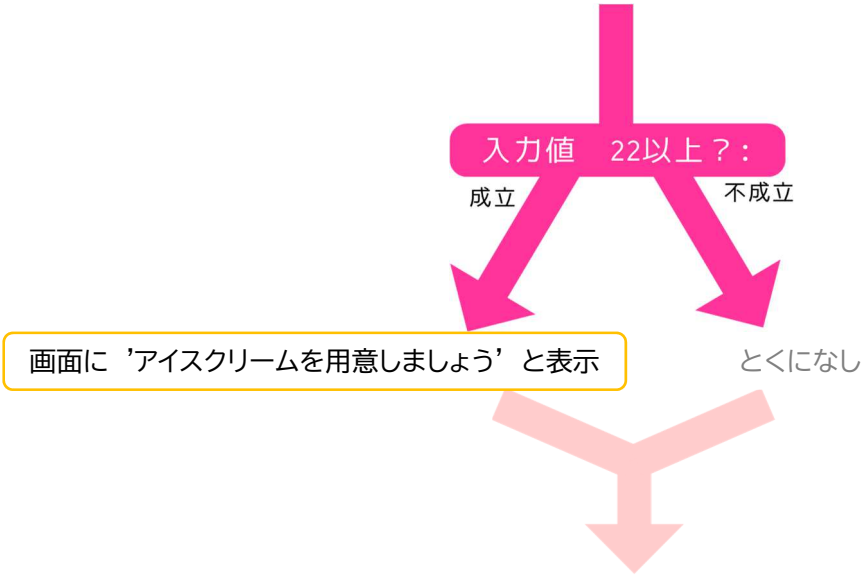
※　条件式に用いるのは、「＝」ではなく「==」、「=<」ではなく「<=」です。間違えないようにしましょう。



「もし、～という条件が成立するならば･･･させる」処理

では、if 文の使い方を踏まえて、次の動作をするプログラムを作ってみましょう。

「もし、入力された値が 22 以上という条件が成立するなら、
画面に'アイスクリームを用意しましょう'と表示させる。」



if 文の働きを示したビデオで見て、if 文による条件分岐のイメージをもってください。

if 文

例題 15　test15.py	
1	temp␣=␣int(input('気温を入力してください'))
2	if␣temp␣>=␣22:
3	␣␣␣␣print('アイスクリームを用意しましょう')

実行結果　例(緑の文字は入力例)

```
気温を入力してください 22
アイスクリームを用意しましょう
```

成立のときの処理も、不成立のときの処理もできる if 文

if 文は、条件式成立のときに処理をさせるだけでなく、不成立のときに処理をさせることもできます。さらに、別の条件式が成立したときに処理をさせることもできます。

if 文

もし、

- ✧ 条件式1が成立するならば処理 A をさせ、
- ✧ 別の条件式2が成立するならば処理 B をさせ、
- ✧ いずれの条件も不成立ならば処理 C をさせます。

使い方

```
if　条件式 1：
␣␣␣␣条件式 1 が成立したときの処理 A
elif　条件式 2:
␣␣␣␣条件式 2 が成立したときの処理 B
else:
␣␣␣␣いずれの条件式も不成立のときの処理 C
処理 D
```

（使用例）

```
if x==1:
␣␣␣␣print('x は 1 です')
elif x==2:
␣␣␣␣print('x は 2 です')
␣␣␣␣print('変数 x の値は 2 です')
else:
␣␣␣␣print('x は 1 でも 2 でもありません')
print('if 文が終わった後の処理です')
```

if 文の行末にコロン:を記述するだけでなく、elif の行末にも、else の後ろのコロン:を忘れずに記述してください（elif はちょっと変わったスペルですので注意してください）。

また、条件によって処理する命令の数は、多くても少なくても構いませんが、字下げ（インデント）をして記述する必要があります。

字下げ（インデント）しないで記述すると if 文の処理ではなくなります。if 文が終わった後に実行されます（処理 D）。

※ elif で始まる別条件を複数設けることもできます。
また、if だけのプログラムでも、if と else だけ、あるいは if と elif だけのプログラムにしも構いません。どんな条件で処理をしたいかに応じて選んでください。

次に示す例題 16 のプログラムを作成してください。

例題 16　test16.py

```
1  temp␣=␣int(input('気温を入力してください'))
2  if␣temp␣>=␣22:␣␣␣␣#␣もし temp が 22 以上なら次の命令を実行
3  ␣␣␣␣print('アイスクリームを用意しましょう')
4  elif␣temp␣<=␣15:␣␣␣␣#␣もし temp が 15 以下なら次の命令を実行
5  ␣␣␣␣print('鍋物を用意しましょう')
6  else:␣␣␣␣#␣そうでないなら次の命令を実行
7  ␣␣␣␣print('特にありません')
8  print('if 文が終わった後の処理です')
```

※ #で始まる箇所をコメント文といいます。
#から右は、プログラムとして処理されなくなります。説明を書くときや、実行させないようにしたいときに使うことができます。# は日本語入力がオンでないときの文字(半角)です。
コメント文は、読みやすくするために、#と半角のスペース 1 個で書き出すことをお勧めします。【PEP8】 P.22

実行結果　例(緑の文字は入力例)

```
気温を入力してください 22
アイスクリームを用意しましょう
if 文が終わった後の処理です
```

```
気温を入力してください 18
特にありません
if 文が終わった後の処理です
```

それでは、これまで学んだ内容をもとに次の課題にチャレンジしてください。

チャレンジ　test17.py

次の条件で動作するプログラムを作成したいと思います。

- ✧ 気温 15℃以下のとき「鍋物を用意しましょう」と表示させ、
- ✧ 気温 22℃以上のとき「アイスクリームを用意しましょう」と表示させ、
  さらに気温 30℃以上のとき「かき氷も用意しましょう」も表示させる
- ✧ それ以外は、「特にありません」と表示させる

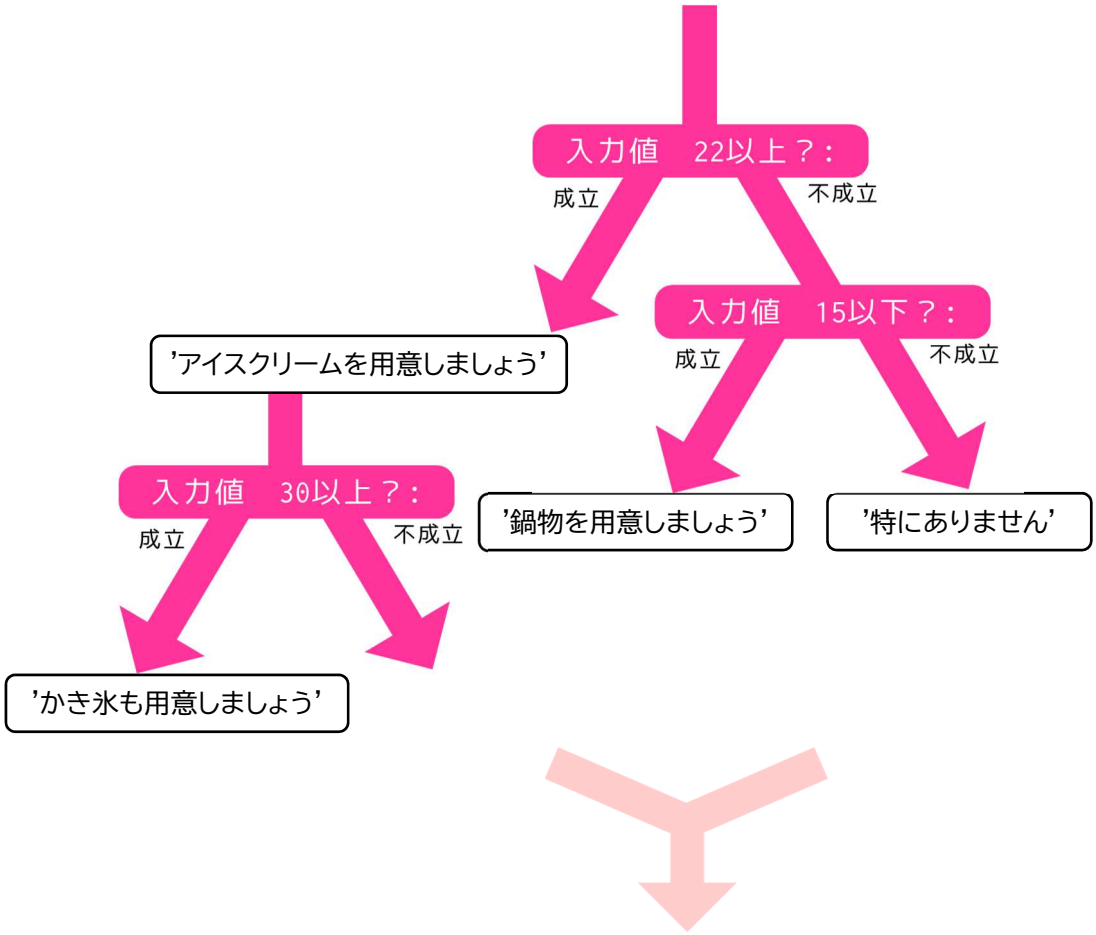
条件に合うように、プログラムのオレンジ色の空欄 　 を埋めたプログラムを作成し、動作を確認してください。

```
1  temp␣=␣int(input('気温を入力してください'))
2  if␣temp␣>=␣  :
3  ␣␣␣␣print("                                  ")
4  ␣␣␣␣if␣temp␣>=␣30:␣␣␣␣#␣さらにもし temp が～なら
5  ␣␣␣␣␣␣␣␣print('かき氷も用意しましょう')
6  elif␣temp␣<=␣  :
7  ␣␣␣␣print('鍋物を用意しましょう')
8  else:
9  ␣␣␣␣print('特にありません')
```

実行結果　例(緑の文字は入力例)

```
気温を入力してください 31
アイスクリームを用意しましょう
かき氷も用意しましょう
```

```
気温を入力してください 10
鍋物を用意しましょう
```



複数の条件分岐

test17.py のように、if 文の条件分岐の処理として if 文を使うこともできます。つまり、2 段階の if 文もできるわけです(もっと複数段階の if 文による条件分岐もできます)。

ただし、条件分岐を並べる順番には、注意が必要です。

たとえば、先に 30 以上という条件式を立て、その後に 22 以上という続く条件式を立てると、30 以上かつ 22 以上のときに print('22 度以上')が処理されることになるので、実質的には 30 以上のときにしか表示されないことになってしまいます。

```
1  temp = int(input('入力してください'))
2  if temp >= 30:
3      print('30 度以上')
4      if temp >= 22:    # temp が 30 以上でさらに 22 以上？なら処理
5          print('22 度以上')
```

ここまでの学習の振り返り

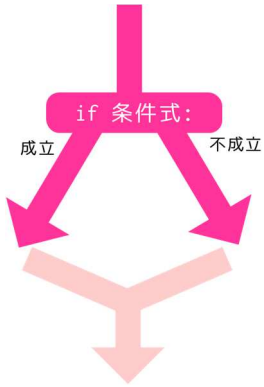
ここまでの学習におけるポイントを確認しましょう。

- `if` 文は、条件式成立のときに処理をさせるだけでなく、不成立のときに処理をさせることもできる
  さらに、別の条件式が成立したときに処理をさせることもできる

- `if` 文は、条件式、コロン`:` を記述する
  条件式が成立(必要に応じて不成立)するときに実行したい処理は、半角スペース 4 つ分字下げ(インデント)して記述する

```
if　条件式 1:
␣␣␣␣条件式 1 が成立したときの処理 A
elif　条件式 2:
␣␣␣␣条件式 2 が成立したときの処理 B
else:
␣␣␣␣いずれの条件式も不成立のときの処理 C
```

- `if` 文に用いる条件式は、代入式ではない(`a=10` ではなく `a==10`)
- 複数段階の `if` 文にすることもできるが、並べる順番に注意する



確認クイズ3

次の QR コードには、ここまで学んだプログラムに関するクイズを載せています。スマホで QR コードを読み、確認クイズにより自身のプログラミングに関する理解度をたしかめてください。

次の目標に到達できましたか？

- 条件分岐のあるプログラムを作成できる。

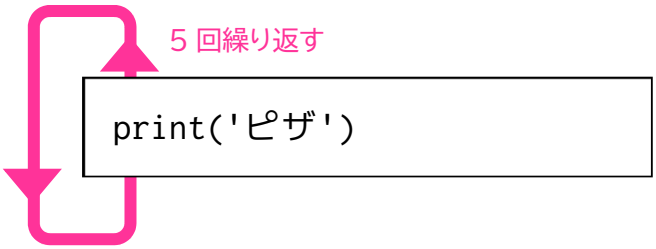
7　繰り返し処理（for 文）

次のような実行結果を得たいとき、どのようなプログラムを記述しますか？

```
ピザ
ピザ
ピザ
ピザ
```

```
1  print('ピザ')
2  print('ピザ')
3  print('ピザ')
4  print('ピザ')
5  print('ピザ')
```

とすればいいですね。
でも、繰り返し命令を使うと、もっと短いプログラムにすることができます。



繰り返し処理は、for という命令を使って次のように書くことができます。

```
1  for count in range(5):
2  ␣␣␣␣print('ピザ')
```

このような繰り返し処理を for 文といいます。キー入力が減るので楽ですし、見やすく便利です。

for 文では繰り返し対象の処理を、半角スペース 4 つ分字下げ（インデント）するところがポイントです。

繰り返し処理は、ぐるぐる回るイメージです。
次のビデオで見て、for 文による繰り返し処理のイメージをつかんでください。

for 文 1

次に示す例題18のプログラムを作成し、実行してみましょう。

例題 18　test18.py

```
1  for␣count␣in␣range(5):
3  ␣␣␣␣print('ピザ')
```

for 文

処理 A を（range()を使えば指定回数分）繰返します。

使い方

```
for 変数 in range(繰返す回数):
␣␣␣␣処理 A
処理 B
```

(使用例)

```
for x in range(3):
␣␣␣␣print('アメンボ赤いな、あいうえお')
␣␣␣␣print('柿の木栗の木、かきくけこ')
print('for 文が終わった後の処理です')
```

(実行結果)

```
アメンボ赤いな、あいうえお
柿の木栗の木、かきくけこ
アメンボ赤いな、あいうえお
柿の木栗の木、かきくけこ
アメンボ赤いな、あいうえお
柿の木栗の木、かきくけこ
for 文が終わった後の処理です
```

字下げ(インデント)しないで記述すると for 文の処理ではなくなります。for 文が終わった後に実行されます(処理 B)。

例題18を次のように修正してみましょう。

例題 18　test18.py

```
1  for␣count␣in␣range(5):
2  ␣␣␣␣print('飛んで')
3  print('まわる')
```

print('飛んで') が 5 回繰り返され、その後に print('まわる') 実行されます。

使い方がわかってきたところで、得点を 5 回尋ねるプログラムを作成し、実行してみましょう。

例題 19　test19.py

```
1  goukei␣=␣0
2  for␣count␣in␣range(5):
3  ␣␣␣␣tokuten␣=␣int(input('得点を入力してください'))
4  ␣␣␣␣goukei␣+=␣tokuten
5  print('合計',␣goukei)
6  print('平均',␣goukei␣/␣5)
```

実行結果　例(緑の文字は入力例)

```
得点を入力してください 100
得点を入力してください 90
得点を入力してください 100
得点を入力してください 50
得点を入力してください 70
合計 410
平均 82.0
```

for 文は複数処理の繰り返しもできる

for 文で繰り返す処理は複数行あっても構いません(例題19では 2 行分を繰り返しました)。for 文が繰り返す範囲は、for 文の次から字下げ(インデント)が終わるところまでです。

変数に値を追加する演算子

例題19では、値を追加する代入演算子 `+=` を使っています。
入力された値を 5 回追加する処理を繰り返すことで合計を出しているわけです。

代入演算子 +=

追加する代入演算子です。

使い方
変数 x += 値 変数 x += 変数 y

（使用例）

```
x = 0
x += 1    # x に 1 が追加される
print(x)  # 1 が表示される

y = 100
x += y    # x に 1 が追加される
print(x)  # 1 が表示される
```

（実行結果）

```
1
101
```

さて、ここで、for 文で使っている変数にも目を向けてみましょう。繰り返しによって変数はどのように変化しているのでしょうか。

次に示す例題 20 のプログラムを通して確認します。

例題 20　test20.py

```
1  for␣count␣in␣range(5):
2  ␣␣␣␣print(count)
```

実行結果

```
0
1
2
3
4
```

for 文と変数の関係を示したビデオで見て、for 文による変数の変化を確認してください。

for 文 2

5 回の繰り返すなかで、変数 count は、0 から 1、2、3、4 と変化します。変数は、0 から繰り返し処理のたびに 1 ずつ増え（回数-1）になります。

1 から 2、3、4、5 と変化するわけではありません。通常の数え方と異なりますので、間違えないように注意してください。

次に示す例題 21のプログラムを作成してください。

例題 21　test21.py

```
1  for␣count␣in␣range(10):
2  ␣␣␣␣if␣ (count␣%␣2) ␣==␣0:
3  ␣␣␣␣␣␣␣␣print(count,␣'偶数')
```

実行結果

```
0 偶数
2 偶数
4 偶数
6 偶数
8 偶数
```

例題21の 2 行目に出てくる　count␣%␣2 の　%　は余りの計算をする演算子です。

演算子	意味	使用例	使用例の結果
x % y	x を y で割った余り	10 % 4	2

つまり、count␣%␣2　は、count 変数を 2 で割った余りという意味になります。

if␣ (count␣%␣2) ␣==␣0:　は、余りが　0　かを判定する条件式ですので、count 変数が偶数なら成立し、if 文に対応した処理(print(count,␣'偶数'))をします。

それでは、これまで学んだ内容をもとに次の課題にチャレンジしてください。

チャレンジ　test22.py

例題　test21.py は、0 から 9 までの数字のうち、偶数しか表示できませんでした。
奇数も表示するようにプログラムの修正をしてください。

```
0 偶数
1 奇数
2 偶数
3 奇数
4 偶数
5 奇数
6 偶数
7 奇数
8 偶数
9 奇数
```

for 文は「繰り返し処理」を繰り返すこともできる

for 文は、「繰り返し処理(for 文による処理)」を繰り返すこともできます。

for 文で「for 文のある処理」を繰り返す様子を示したビデオで見て、for 文による2重繰り返しの動きを確認してください。

for 文 3

例題 23　test23.py

```
1  for␣cnt1␣in␣range(3):
2  ␣␣␣␣for␣cnt2␣in␣range(3):
3  ␣␣␣␣␣␣␣␣print('cnt1=',␣cnt1,␣'cnt2=',␣cnt2)
```

実行結果

```
cnt1= 0 cnt2= 0
cnt1= 0 cnt2= 1
cnt1= 0 cnt2= 2
cnt1= 1 cnt2= 0
cnt1= 1 cnt2= 1
cnt1= 1 cnt2= 2
cnt1= 2 cnt2= 0
cnt1= 2 cnt2= 1
cnt1= 2 cnt2= 2
```

それでは、これまで学んだ内容をもとに次の課題にチャレンジしてください。

チャレンジ　test24.py

例題　test23.py をもとに、for 文を用いて、次の表示をするプログラムを作成してください。

```
13 時 0 分
13 時 1 分
13 時 2 分
13 時 3 分
14 時 0 分
14 時 1 分
14 時 2 分
14 時 3 分
15 時 0 分
15 時 1 分
15 時 2 分
15 時 3 分
16 時 0 分
16 時 1 分
16 時 2 分
16 時 3 分
```

ここまでの学習の振り返り

ここまでの学習におけるポイントを確認しましょう。

- for 文で処理を繰返すことができる
- for 文は、次のように記述する(コロン:忘れず)
  繰返したい処理(A、B)は半角スペース 4 つ分字下げ(インデント)をする

```
for␣変数␣in␣range(回数):
␣␣␣␣処理 A
␣␣␣␣処理 B
処理 C
```

- for 文終了後に実行する処理(C)は字下げをしない
- for␣変数␣in␣range(回数):としたとき、
  変数は、0 から繰り返し処理のたびに 1 ずつ増え (回数-1) になる

確認クイズ4

次の QR コードには、ここまで学んだプログラムに関するクイズを載せています。スマホで QR コードを読み、確認クイズにより自身のプログラミングに関する理解度をたしかめてください。

次の目標に到達できましたか？

- 繰り返し処理のプログラムを作成できる。

8　変数を一括処理する配列

複数の値を 1 つの変数名で管理する仕組みに、配列があります。
（標準のPythonで使える配列はリストと呼ばれます。）

配列

複数の値を 1 つの変数名で管理する仕組みです。
配列の名付けルールは、変数と同じです。

配列も、変数と同じように整数や小数といった数値や文字列を格納できますが、変数と異なるのは、値をカンマ(,)で区切り、全体を[]で囲んで格納する必要があることです。

また、配列の参照や代入は、配列名[インデックス番号]としておこないます。インデックス番号は、0 からの通し番号で、(初期化で格納した値数-1)までとなります。

使い方

```
（初期化・・・配列に扱いたいデータ数分の値を格納する）
配列名 = [ 値, 値,・・・, 値]

（個々の参照・代入）
配列名[インデックス番号]
```

(使用例)

```
b_year = [1853, 1864, 1840]
stuff_name = ['北里', '津田', '渋沢']
print(b_year[0])
print(stuff_name[0])
print(stuff_name[2])
print(stuff_name[1])
```

(実行結果)

```
1853
北里
渋沢
津田
```

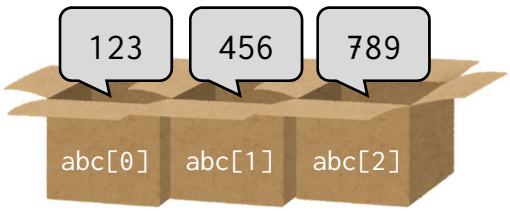
たとえば、abc という名前の配列に整数3つを格納(初期化)する場合、次のような記述となります。

```
abc␣=␣[123,␣456,␣789]
```

この式により、abc[0]、 abc[1]、 abc[2] が用意され

- abc[0]に 123 が、
- abc[1]に 456 が、
- abc[2]に 789 がそれぞれ格納されます。

[0]番目から順に[2]まで3つ用意され、
用意された配列への参照や代入ができるようになります。



配列は、連なった箱や下駄箱をイメージするといいでしょう。

注意が必要なのは、配列名[1]からではなく、配列名[0]から用意・代入されるということです。

それでは、3つの整数をまとめて管理する days と名付けた配列のプログラムを通して、配列の仕組みと動作を確認してください。

例題 25　test25.py

```
1  days␣=␣[29,␣31,␣30]
2  print('2020 年 2 月は、',␣days[0],␣'日あります')
3  print('2020 年 3 月は、',␣days[1],␣'日あります')
4  print('2020 年 4 月は、',␣days[2],␣'日あります')
5  print(days)
```

実行結果

```
2020 年 2 月は、  29 日あります
2020 年 3 月は、  31 日あります
2020 年 4 月は、  30 日あります
[29, 31, 30]
```

配列の初期化、配列のカタチ

配列名[インデックス番号]で参照や代入をする前に、配列の初期化が必要です。

例題 25 では、1 行目で配列 days の初期化をしました(days = [29, 31, 30])。
格納されていない配列を参照するとエラーになります(days[3]は入っていません)。

なお、print(配列名)　とすると、その配列に格納されている値がカンマ(,)で区切られ、[]で囲まれたかたちで表示されます。print(配列名)　あるいは　print(変数名)　とすることにより、配列なのか変数なのか、違いを知ることができるだけでなく、配列に入っている値やその数を確認することができます。

それでは、これまで学んだ内容をもとに次の課題にチャレンジしてください。

チャレンジ　test26.py

次のように表示されるよう、　　を埋めたプログラムを作成してください。

実行結果　例(緑の文字は入力例)

```
2020 年  1 月は  31 日ある
2020 年  2 月は  29 日ある
2020 年  3 月は  31 日ある
2020 年  4 月は  30 日ある
2020 年  5 月は  31 日ある
2020 年  6 月は  30 日ある
2020 年  7 月は  31 日ある
2020 年  8 月は  31 日ある
2020 年  9 月は  30 日ある
2020 年  10 月は  31 日ある
2020 年  11 月は  30 日ある
2020 年  12 月は  31 日ある
何月ですか？3
2020 年  3 月は  31 日ある
```

プログラム

```
1  days = [                                                    ]
2  for month in range(12):
3      print('2020 年', month + 1, '月は', days[month], '日ある')
4  month = int(input('何月ですか？'))
5  print('2020 年', ,'月は', days[ ], '日ある')
```

次に示す例題 27のプログラムを作成してください。

例題 27　test27.py

```
1  stuff_name = ['', '', '', '', '']
2  for num in range(5):
3      stuff_name[num] = input('名前を入れてください')
4  n = int(input('何番目の方の名前が知りたいですか？'))
5  print(n, '番目は', stuff_name[n], 'さんです')
```

実行結果　例(緑の文字は入力例)

```
名前を入れてくださいトーマス
名前を入れてくださいパーシー
名前を入れてくださいジェームス
名前を入れてくださいゴードン
名前を入れてくださいスペンサー
何番目の方の名前が知りたいですか？3
3 番目は ゴードン さんです
```

配列には「初期化」が必要

配列名[インデックス番号]で参照や代入をする前に、あらかじめ配列を初期化しておく必要があります。

初期化とは扱いたいデータ数分の値を配列に格納することです。

例題 27 の 1 行目 `stuff_name = ['', '', '', '', '']` が初期化の部分になります。空のデータの代入でもよいので配列に値を代入します。こうすることで、代入したデータ数分の配列が用意されるのです。

この 1 行目の配列の初期化がないと、値を代入するところでエラーになってしまいます。

ためしに、例題 27 の 1 行目の先頭に `#` を付けて（コメント文となり、実行されなくなります）、プログラムを実行させてみましょう。

```
1    # stuff_name = ['', '', '', '', '']
```

実行結果　例（緑の文字は入力例）

```
名前を入れてくださいトーマス
Traceback (most recent call last):
  File "/home/pi/Documents/test27.py", line 3, in <module>
    stuff_name[num] = input('名前を入れてください')
NameError: name 'stuff_name' is not defined
```

代入しようとした配列 stuff_name は定義されていないという理由でのエラー（NameError）が出ました。 初期化が必要であることが確認できたと思います。

配列には、 `#` を削除して元に戻してください。

配列名[インデックス番号]で参照や代入をする前に、あらかじめ配列を初期化しておくことが必要なのです。

さて、Python には、はじめから組み込まれている便利な関数があります。
その中から、3 つの関数(合計を返す sum()関数、最大値を返す max()関数、最小値を返す min()関数)を使うことにします。

max()関数

引数として渡した配列の最大の要素を返します。

使い方

```
max(配列)
```

(使用例)

```
abc = [1, 22, 333]
print(max(abc))
```

(出力結果)

```
333
```

min()関数

引数として渡した配列の最小の要素を返します。

使い方

```
min(配列)
```

(使用例)

```
abc = [1, 22, 333]
print(min(abc))
```

(出力結果)

```
1
```

sum()関数

引数として渡した配列(数値)の合計を返します。

使い方

```
sum(配列)
```

(使用例)

```
abc = [1, 22, 333]
print(sum(abc))
```

(出力結果)

```
356
```

len()関数

引数として渡した配列の要素数を返します。

使い方

```
len(配列)
```

(使用例)

```
abc = [1, 22, 333]
print(len(abc))
```

(出力結果)

```
3
```

次に示す例題 28のプログラムを作成してください。

例題 28　test28.py

```
1  tokuten␣=␣[0,␣0,␣0,␣0,␣0]
2  for␣count␣in␣range(5):
3  ␣␣␣␣print(str(count␣+␣1)␣+␣'人目')
4  ␣␣␣␣tokuten[count]␣=␣int(input('得点を入力してください'))
5  print('合計',␣sum(tokuten))
6  print('最大値',␣max(tokuten),␣'最小値',␣min(tokuten))
```

実行結果　例(緑の文字は入力例)

```
1 人目
得点を入力してください 100
2 人目
得点を入力してください 90
3 人目
得点を入力してください 100
4 人目
得点を入力してください 50
5 人目
得点を入力してください 70
合計 410
最大値 100 最小値 50
```

それでは、これまで学んだ内容をもとにこのテキスト最後の課題にチャレンジしてください。

チャレンジ　test29.py

7 人の得点を尋ね、
合計が 700 なら、 ’全員満点です’ と表示し、
最高点と最低点の差が 50 より大きければ ’個人差が大きいです’ と表示するプログラムを作成してください。

ここまでの学習の振り返り

ここまでの学習におけるポイントを確認しましょう。

- 配列の名前は、変数と同様のルールでつける
- 配列名[インデックス番号]で参照や代入をする前に
  配列名 ＝[値, 値,･･･, 値] で配列の初期化をする
  （扱いたいデータ数分の値を配列に格納すること）
- 配列名[インデックス番号] で参照･代入できる
  インデックス番号は、0 から数えた値（つまり、配列名[0] から参照･代入できる）

確認クイズ5

次の QR コードには、ここまで学んだプログラムに関するクイズを載せています。スマホで QR コードを読み、確認クイズにより自身のプログラミングに関する理解度をたしかめてください。

次の目標に到達できましたか？

- 配列(リスト)を使ったプログラムを作成できる。

9 まとめ

このテキストでは、学習目標達成を目指して、プログラミングの学習を進めてきました。

目標に到達することができたでしょうか？

この教材の学習目標
値や変数、配列(リスト)を用いた順次処理や条件分岐、繰り返し処理を組み合わせた Python のプログラムを作成できる。 1. Python のプログラムを入力し、動作確認できる。 2. 順次処理のプログラムを作成できる。 3. 変数について説明できる。 4. 変数を使ったプログラムを作成できる。 5. 条件分岐のあるプログラムを作成できる。 6. 繰り返し処理のプログラムを作成できる。 7. 配列(リスト)を使ったプログラムを作成できる。

10 おわりに

29 の課題を通して、Python のプログラミングを学んできましたが、いかがでしたでしょうか？

本書で学んだ内容とプログラム開発環境でちょっとしたプログラムならできそうだと思えたなら、ぜひ作ってみてください。さらにプログラミングの理解が深まるとともに、新しいプログラムに対しても、やっていけそうな見通しを持てるようになるかと思います。

Python のプログラムは、もっともっと発展させることができます。例えば、事務処理の自動化、機械の制御、画像処理などです（さわりだけではありますが、次のページに参考プログラムを載せています）。

プログラミング技術を実際の仕事で活かせるならば、コンピュータはあなたの相棒になります。周りをよく見まわしてください。プログラミング技術を活かせる場面がたくさんあるのではないでしょうか。ここで学んだプログラミング技術を知識にとどめず活用・応用し、実践力にしてください。プログラミングを楽しさを知り、活かせるようになり、ご活躍されることを心より願っております。

最後まで読んでいただきありがとうございました。

豆知識　Python でできる様々な処理

Python のプログラミングにより、様々な処理を実現できます。例を示します。
このなかに作成したいプログラムがあったら、ぜひチャレンジしてみてください。

※ ただし、ライブラリが別途必要になります。ライブラリのインストール方法は先生に尋ねてください

事務処理の自動化（例： ファイルの名前を変更するプログラム）

```
1  import␣os
2
3  path1␣=␣'./レポート.docx'   # 変更前のファイル名
4  path2␣=␣'./報告書.docx'     # 変更後のファイル名
5  os.rename(path1,␣path2)     # ファイル名を変更する
```

機械の制御（例： Raspberry Pi で GPIO に接続した LED を点滅させるプログラム）

```
1  import␣RPi.GPIO␣as␣GPIO
2  import␣time
3
4  GPIO.setmode(GPIO.BCM)
5  GPIO.setup(15,␣GPIO.OUT)     # LED を接続した GPIO15 を OUTPUT に設定
6  while␣True:                  # ずっと繰り返す
7  ␣␣␣␣GPIO.output(15,␣True)   # GPIO15（LED）を点灯
8  ␣␣␣␣time.sleep(2)            # 2 秒待つ
9  ␣␣␣␣GPIO.output(15,␣False)  # GPIO15（LED）を消灯
10 ␣␣␣␣time.sleep(2)            # 2 秒待つ
11 GPIO.cleanup()
```

画像認識（例： カメラの画像を表示させるプログラム）

```
1  import␣cv2
2
3  cap␣=␣cv2.VideoCapture(0)                  # 0 はカメラのデバイス番号
4  while␣True:                                # ずっと繰り返す
5  ␣␣␣␣ret,␣frame␣=␣cap.read()               # カメラから画像を読み込む
6  ␣␣␣␣cv2.imshow('camera capture',␣frame)    # 画像をウィンドウに表示する
7  ␣␣␣␣k␣=␣cv2.waitKey(1)                     # キー入力を 1m 秒待つ
8  ␣␣␣␣if␣k␣==␣27:                            # [ESC]キーを押したら、
9  ␣␣␣␣␣␣␣␣break                              # 繰り返しから抜ける
10 cap.release()
11 cv2.destroyAllWindows()                    # ウィンドウを消す
```

Python では、ほかにも機械学習、Web アプリ開発などなど様々なプログラミングができます。

はじめてのプログラミング　～Python 編～

著者　　● 松本和重
イラスト　　● いらすとや
コンテンツ制作　● 松本和重

（参考文献）

- 長谷川　聡・山住富也（1997）プログラミング教育と学習者のイメージ形成，名古屋文理短期大学紀要 22(0), pp.9-14
- 長谷川 聡・山住富也（1998）プログラミング教育と学習者のイメージ形成（その 2），名古屋文理短期大学紀要 23(0), pp.9-14
- 宮川　治・土肥紳一・今野紀子・高野辰之・小濱隆司（2015）プログラミング演習でのつまずきに関する分析, 日本教育心理学会総会発表論文集 57(0), p.249
- 岡本雅子・村上正行・吉川直人・喜多　一（2013）<実践報告>プログラミングの写経型学習過程を対象としたつまずきの分析とテキスト教材の改善 : 作業の自立的遂行と作業を介した理解のための支援と工夫, 京都大学高等教育研究（19), pp.47-57
- 岡本雅子・喜多　一（2014）プログラミングの「写経型学習」における初学者のつまずきの類型化とその考察, 滋賀大学教育学部附属教育実践総合センター紀要(22), pp.49-53

- Python »（Japanese 3.7.8）Documentation
  https://docs.python.org/ja/3.7/contents.html　（202007 確認）
- Python.org　PEP 8 -- Style Guide for Python Code,
  https://legacy.python.org/dev/peps/pep-0008/　（202007 確認）

はじめてのプログラミング　～Python 編～