高等学校プログラミング教育

Python編

生徒テキスト

**基　本　編**

**Google Colaboratory p.5**

**1**

1.1. PythonとGoogle Colaboratory

1.2. Google Colaboratoryを使ってみよう

1.3. コードセルとテキストセル

1.4. テキストセルを使ってみよう

**数値型と文字列型 p.10**

**2**

1.5. コードセルを使ってみよう

2.1. 数値型

2.2. 文字列型

**変数 p.13**

**3**

2.3. 文字列型と数値型の違い

3.1. 変数を用いた値の記憶

**条件分岐 p.16**

**4**

4.1. if文

4.2. else文

4.3. elif文

4.4. if文の入れ子

**配列 p.25**

**5**

5.1. 配列とは

5.2. 要素の追加

5.3. 要素の削除

5.4. 配列の長さを調べる

**ループ p.29**

**6**

6.1. for文

6.2. 配列の各要素の参照

6.3. while文

**関数 p.33**

**7**

7.1. 関数とは

7.2. 関数の引数

7.3. ローカル変数とグローバル変数

7.4. 関数の戻り値

**グラフィックス p.38**

**8**

**発　展　編**

8.1. Turtle Graphicsのインストール

8.2. 基本的な操作

8.3. 図形を描いてみよう

**幾何学模様 p.52**

**9**

9.1. 円を使った幾何学模様

**デジタル単語帳 p.56**

**10**

10.1. デジタル単語帳の設計

10.2. 入力を受け取る関数

10.3. ランダムに値を代入してみよう

10.4. 実際にプログラムを記述してみよう

**応用編１（数あてゲームの作成）**

**Hit and Blow p.62**

**11**

**応　用　編**

11.1. 数値を文字列に変換する

11.2. 文字列のn番目を取得する

11.3. in演算子

11.4. 条件式の「かつ」「または」

11.5. Hit and Blowのルール

11.6. Hit and Blowをプログラムで記述する

11.7. もっと使いやすくしてみよう

**応用編２（迷路を作成してゴールを目指そう）**

**迷路を自動生成するには p.69**

**12**

12.1. 二次元配列

12.2. 関数の再帰

12.3. 迷路の生成アルゴリズム

12.4. 迷路の解法アルゴリズム

12.5. 配列からランダムで選ぶ

12.6. 迷路の設計

12.7. 迷路の生成プログラムの作成

12.8. 迷路の解法プログラム

1. Google Colaboratory
   1. PythonとGoogle Colaboratory

　プログラミングをするために使用される、メジャーなプログラミング言語の1つとして**Python**があります。Googleの「**Google Colaboratory」というウェブサイト**を用いると同一画面上でPythonのプログラムを入力・実行ができるため、ここではこのサイトを利用して学習をおこないます。

* 1. Google Colaboratoryを使ってみよう

　Google Colaboratory（<https://colab.research.google.com/>）にアクセスすると、図1.1のような画面が出ます。まず、自身のアカウントにアクセスします。

① 右上の「ログイン」ボタンをクリックします。

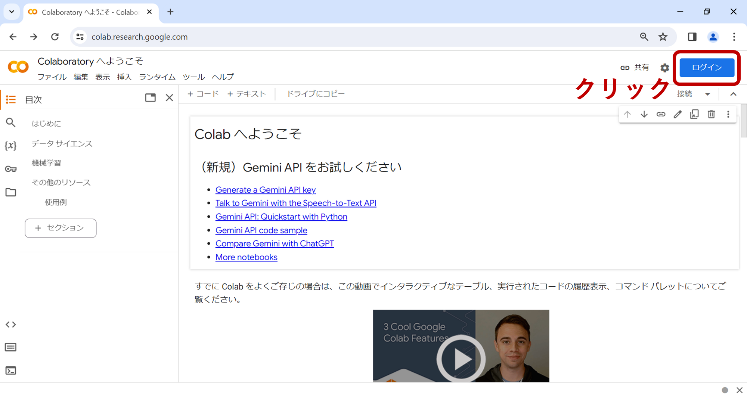


図1.1 Google Colaboratoryの初期画面

② 「メールアドレスまたは電話番号」に自身のメールアドレスを入力し、「次へ」ボタンをクリックします。



図1.2 メールアドレス入力画面

③ 「パスワードを入力」にパスワードを入力し、「次へ」ボタンをクリックします。

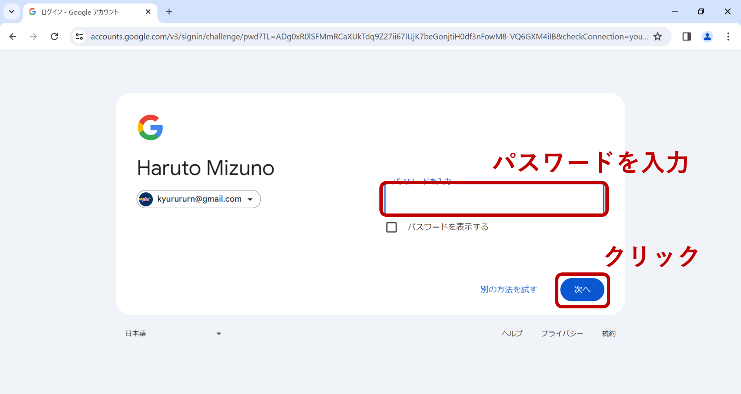


図1.3 パスワード入力画面

　Google Colaboratoryでは、**ノートブック**を用いてPythonプログラムを作成します。ノートブックを新しく作成しましょう。なお、１つのノートブックの中に、複数のPythonプログラムを記述できます。

④ 「ノートブックを新規作成」をクリックします。

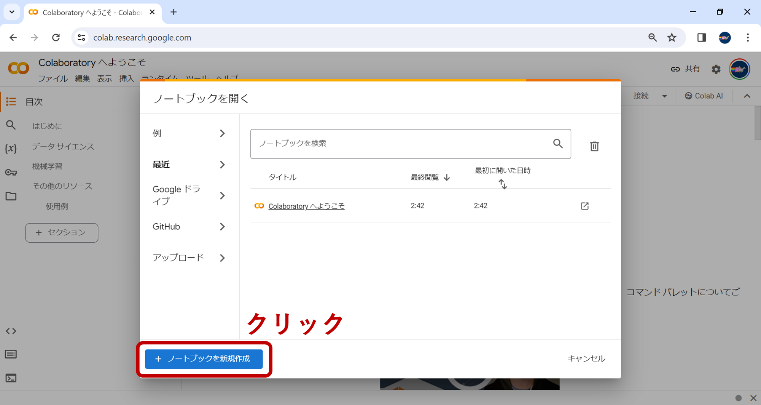


図1.4 ノートブックの新規作成

⑤ 図1.1の画面から新しくノートブックを作成する場合は、「ファイル」を選択し「ノートブック」をクリックします。また、今まで作成したノートブックを使用する場合は、「ファイル」から「ノートブックを開く」をクリックします。



図1.5 ノートブックの新規作成・開く

ノートブックを新規作成すると、図1.6の画面になります。

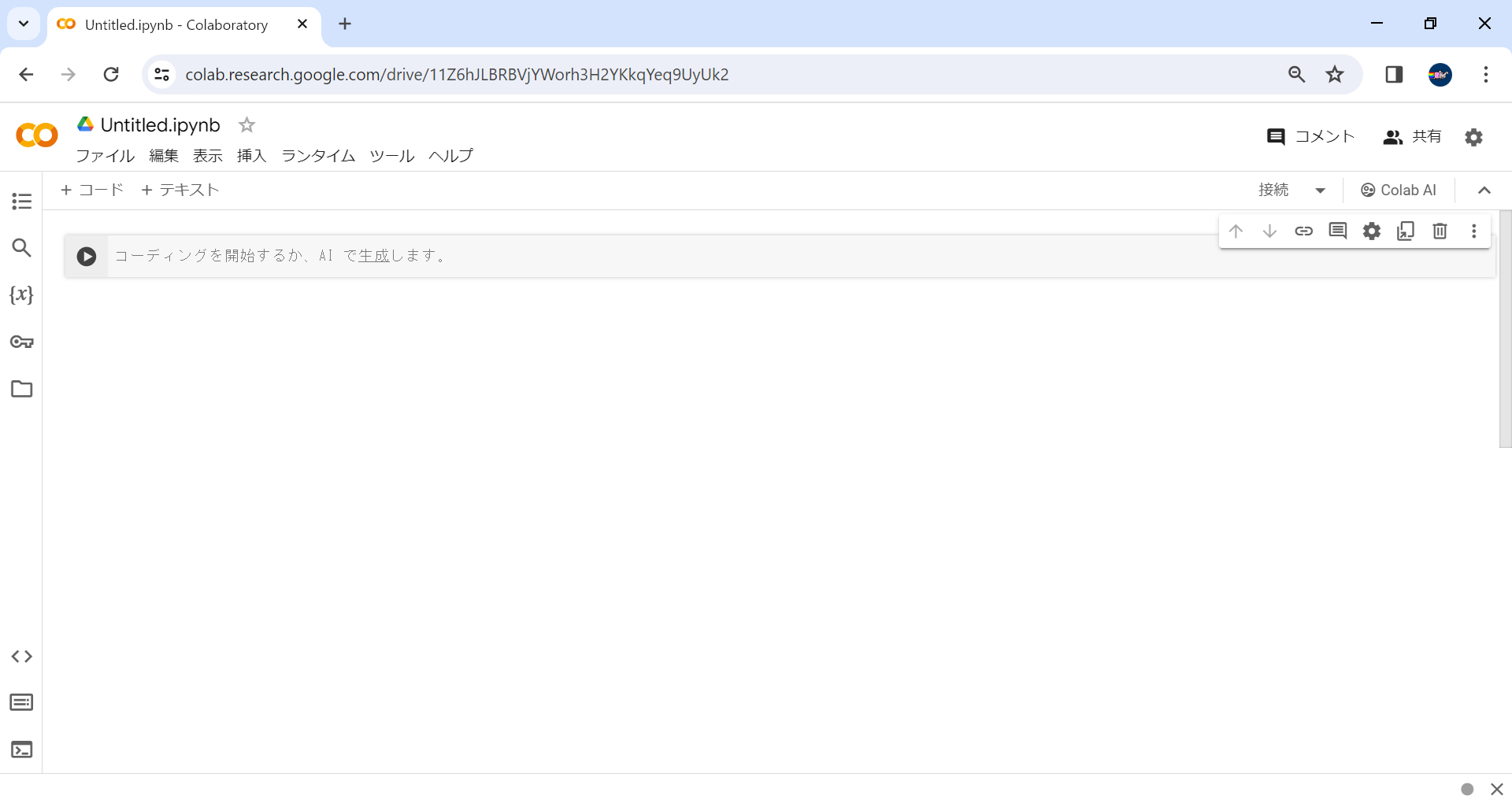


図1.6 新規作成されたノートブック

* 1. コードセルとテキストセル

　Google Colaboratoryでは、**セル**を用いてプログラミングを行います。セルには、**コードセル**と**テキストセル**の2種類があります。コードセルは、Pythonのプログラムを記述する時に、テキストセルは文章（当授業では注意事項のメモや授業の感想等）を記述する時に使用します。左上にある「+ コード」・「+ テキスト」をクリックすると、コードセル・テキストセルが追加されます。また、セルの下側にマウスカーソルを合わせると、「+ コード」・「+ テキスト」が表示されます。それらをクリックしても、コードセル・テキストセルを追加することができます。

　また、セルをクリックするとセルの編集画面が表示されます。セルに文字を入力したり、セルを削除したりすることができます。キーを入力すると文字が入力されます。また、右側に出てくるゴミ箱のマークをクリックすると、セルを削除することができます。

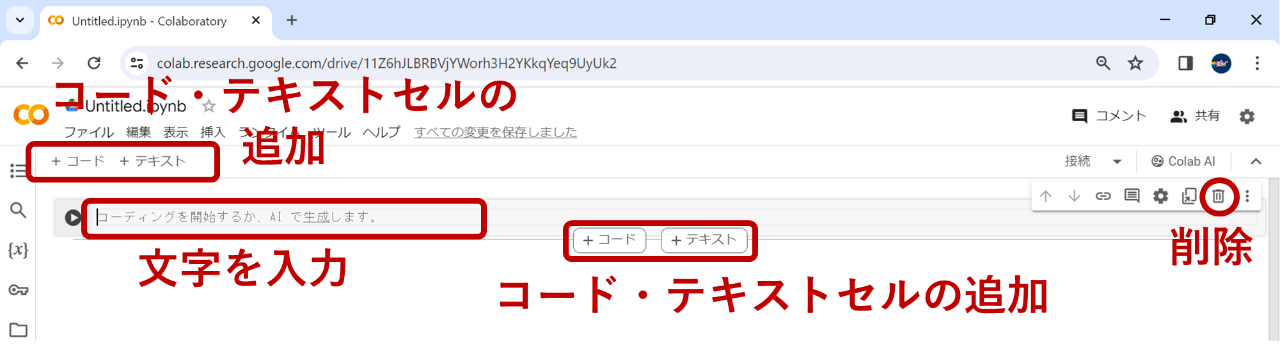


図1.7 セルの追加・編集

* 1. テキストセルを使ってみよう

　テキストセルを追加すると、文字や文章を入力することができます。セルをクリックすると、左側に文字の入力、右側にプレビューが出ます。試しに、テキストセルを追加して好きな文字を入力してみましょう。



図1.8 テキストセルの編集

　Escキーを押すと、入力モードから表示モードへ変更されます。このようにしてテキストセルを使います。



図1.9 テキストセルを表示モードに変更

* 1. コードセルを使ってみよう

　コードセルを追加すると、Pythonのプログラムを記述することができます。

**Pythonのprint関数**

　Pythonのprint関数を用いると、文字を出力することができます。コードセルを追加して、print("Hello Python")と入力をしてみましょう。実行を行う際は、▶をクリックします。



図1.10 print関数による文字の出力

　図1.10のようにprint関数を用いると文字を出力することができます。ダブルクォーテーション（””）の間に入力した文字が出力されます。プログラムを複数行実行する場合は、Enterで改行を行った後に次の行にプログラムを記述する必要があります。また、実行は基本的に上から下の行にかけて行われます。

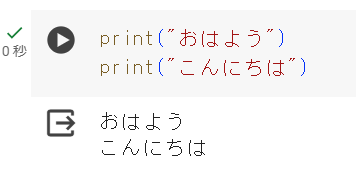


図1.11 複数行のプログラムの実行

**●練習問題1**

　コードセルにprint関数で文字を出力するプログラムを記述して、そのプログラムを解説する説明文をテキストセルに入力してみましょう。

**●解答例1**

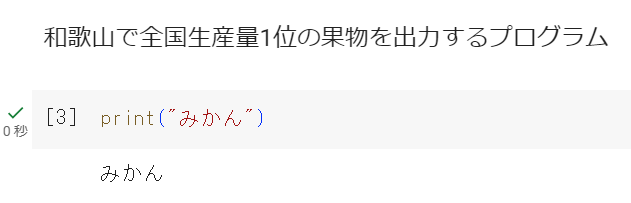


図1.12 解答例1

1. 数値型と文字列型
   1. 数値型

**数値型**は、数をそのまま入力することによって使用します。試しに、コードセルにprint(10)など、print関数の括弧内に数を入れて実行してみましょう。

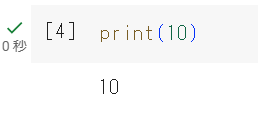


図2.1 数値型の出力

**数値型の演算**

　数値型は、数の計算をそのまま使用することができます。表2.1は、数値型で使用できる演算です。print(1+4)などのようにprint関数の中に式を入れて実行してみましょう。

表2.1 数値型で使用できる演算

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 種類 | 記号 | 機能 | 使用例 | 結果 |
| 足し算 | + | 左辺と右辺を足す | 1 + 3 | 4 |
| 引き算 | - | 左辺から右辺を引く | 2 – 1 | 1 |
| 掛け算 | \* | 左辺と右辺を掛け合わせる | 3 \* 6 | 18 |
| 割り算 | / | 左辺から右辺を割る | 6 / 2 | 3 |
| 余り | % | 左辺から右辺を割った余りを求める | 5 % 3 | 2 |

　計算をする時は、掛け算・割り算・余りを先に計算します。また、print(2\*(1+2))などのように括弧で括ると、括弧内を先に計算します。表2.1の演算を組み合わせて、数値の計算を行ってみましょう。

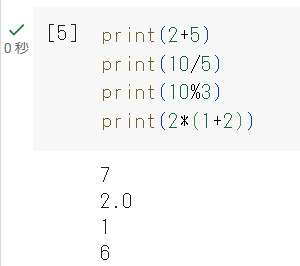


図2.2 数値型の計算

* 1. 文字列型

　文字列型は、ダブルクォーテーションで囲むことによって使用します。コードセルにprint(“Apple”)など、print関数の中にダブルクォーテーションで囲んだ文字を入力して実行してみましょう。

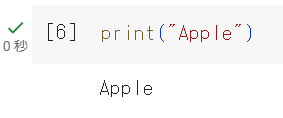


図2.3 文字列の出力

**文字列型の演算**

　文字列型では、数値型と異なる演算を行います。文字列型同士の演算では、+による結合のみ使用できます。数値型のように引き算・掛け算・割り算・余りを用いることはできません。

表2.2 文字列型で使用できる演算

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 種類 | 記号 | 機能 | 使用例 | 結果 |
| 結合 | + | 左辺と右辺を結合する | “Hello” + “World” | “HelloWorld” |

　+による結合を用いて、文字列の結合を行ってみましょう。



図2.4 文字列の結合

* 1. 文字列型と数値型の違い

　① print(1+2)

　② print(“1”+”2”)

　①と②の違いを考えてみましょう。①は数値として演算を行っているため、数の足し算を行い3が出力されます。それに対し、②は文字列として演算を行っているため、結合を行い”12”が出力されます。同じ数字を入力しても、数値型として扱うか文字列型として扱うかで演算結果が異なるため注意が必要です。

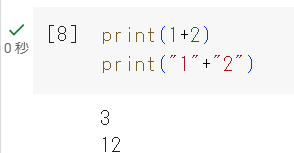


図2.5 文字列型と数値型の演算結果の違い

**●練習問題2**

　和歌山県の人口と面積を検索し、割り算を用いて人口密度を計算するプログラムを記述してみよう。

**●解答例2**

　人口は和歌山県が発表している「和歌山県の推定人口（令和5年4月1日現在）」、面積は和歌山県が作成している「グラフで見る和歌山県　2023年版」を用いている。

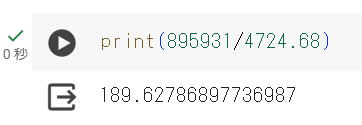


図2.6解答例2

【補足】図2.6では、小数点が１４桁も表示されてしまい非常に見にくいです。intを用いると、小数点を四捨五入することができます。また、文字と数字を混合させて表示する場合は、printの（　）の中に「，」＝カンマを入れます。

例）　print(int(895931/4724.68),”人/平方キロメートル”)

1. 変数
   1. 変数を用いた値の記憶

　変数を用いると、数値や文字列を記憶することができます。「変数名 = 値」をすることで、変数に値を対応させることができます。変数に値を対応させることを**代入**と言います。図3.1は、aを変数としたプログラムです。

① 1行目で変数aに数値10を代入します。変数aに数値10が対応されます。

② 2行目で変数aを出力します。変数aに対応している数値は10です。そのため、10が出力されます。

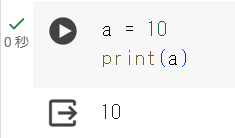


図3.1 変数を用いたプログラム

　変数の代入を、箱を用いて考えることもできます。「a = 10」とすると、aという箱に数値10が入ります。箱aを指定すると、箱の中に入っている数値10が指定されます。

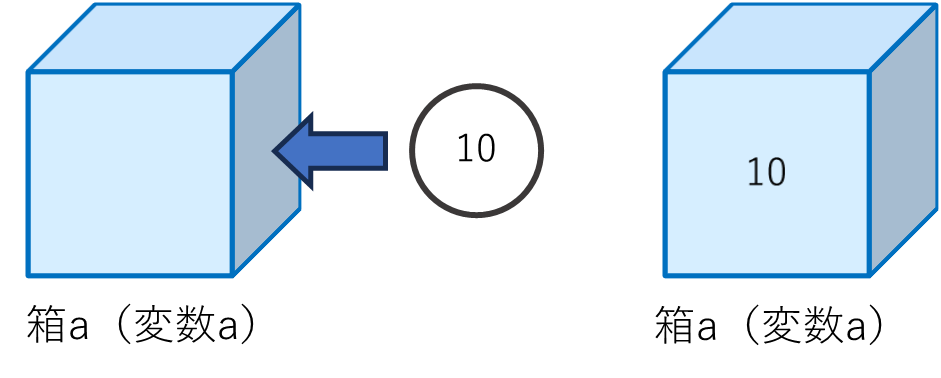


図3.2 箱を用いた変数の考え

　変数には、値を何度でも代入することができます。代入される度に変数の値は上書きされます。図3.3のプログラムでは、

① 1行目で、変数bに数値3を代入します。

② 2行目で、変数bに数値6を代入します。この時、変数bの値は上書きされます。

③ 3行目で、変数bを出力します。最後に代入された数値6が出力されます。

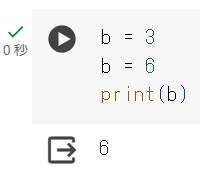


図3.3 変数の上書きを用いたプログラム

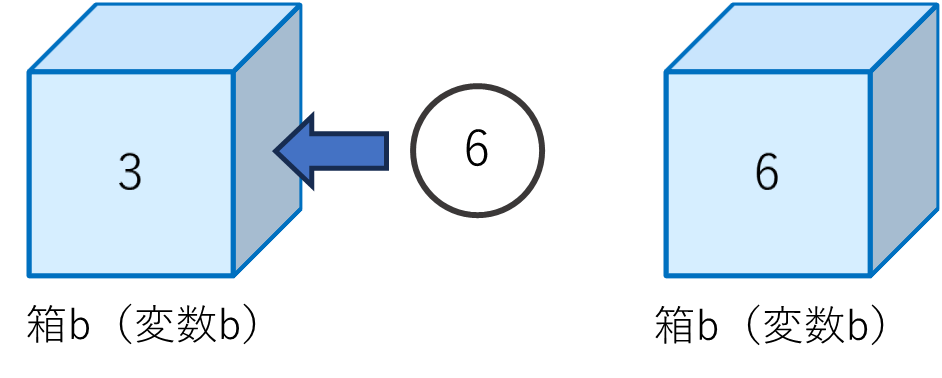


図3.4 箱を用いた変数の上書きの考え

　図3.5のように、変数には数値型と文字列型の両方を代入することができます。

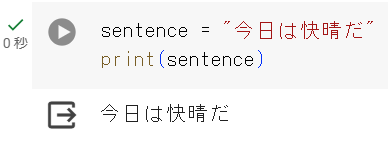


図3.5 変数に文字列を代入

　図3.6のように、変数を用いて演算をすることも可能です。

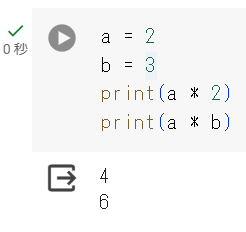


図3.6 変数を用いた演算

　代入をする時に、代入先の変数と同じ変数を、=の右辺に用いることも可能です。この時、右辺から先に計算されます。図3.7のプログラムでは、

① 変数aに数値10を代入します。

② 変数aにa + 1を代入します。a + 1を計算すると11であるため、変数aに数値11が代入されます。

③ 変数aを出力します。

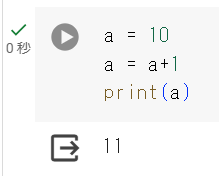


図3.7 代入の際に右辺に同じ変数を用いる

　また、a = a + {数値}というプログラムは、a += {数値}と記述することができます。この代入方法を加算代入と言います。図3.7のa = a+1というプログラムは、a += 1と記述できます。

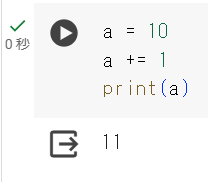


図3.8 加算代入を用いたプログラム

**●練習問題3**

　登校中、太郎君は分速90mで歩きます。太郎君の家から学校までは3600mあります。太郎君の歩く速さを変数speedに、太郎君の家から学校までを変数distanceに代入しましょう。また、何分で学校に着くかを求めるプログラムを作りましょう。

**●解答例3**

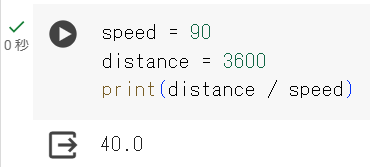


図3.9 解答例3

1. 条件分岐

　Pythonでは、条件によって特定のプログラムを実行するかしないかを選択することができます。

* 1. if文

　if文は、条件に合致をしていたらプログラムを実行します。日本語で表現すると、「もし～ならば」となります。if文は図4.1のような形式で記述します。

① 「if」と入力をします。

② スペースで空白を入れてから条件文を記述します。

③ 「:」と入力をして改行を行います。

④ 改行を行うと、自動で次の行に空白が入ります。これを**インデント**といいます。インデントを手動で行う場合は「tab」キーを押します。

⑤ インデントされている箇所に条件が正しい時のプログラムを記述します。

条件文が正しい時に、条件が正しい時に実行するプログラムが実行されます。

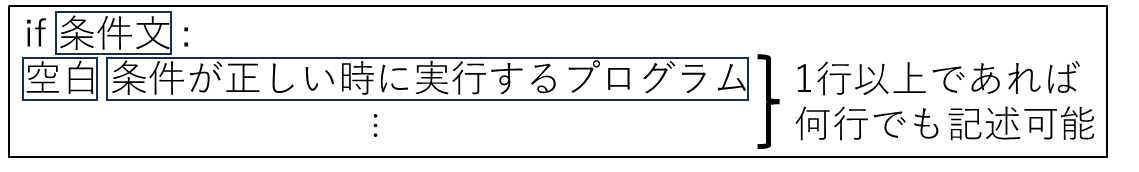


図4.1 if文の形式

　if文を用いると、図4.2のようなプログラムを記述することができます。

① 1行目で、変数aに10を代入します。

② 2行目のif文に記述した条件文が正しいかどうかを確かめます。「a > 5」は「変数aが5よりも大きい」という条件文を示します。変数aは10であるため、条件文が正しいです。

③ 条件が正しいため、条件が正しい時に実行するプログラムが実行されます。

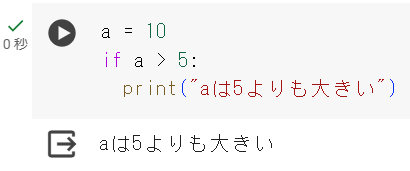


図4.2 条件文を用いたプログラム

　また、条件に当てはまらない時は、図4.3のようにプログラムは実行されません。

① 1行目で、変数aに1を代入します。

② 今回、変数aは1であるため、変数aは5よりも大きくありません。条件文は正しくないです。

③ 条件が正しい時に実行するプログラムは実行されません。



図4.3 条件が正しくない時のプログラム（何も出力されない）

　if文は、**インデントされている箇所＝文頭にスペースのある行のみ**「条件に当てはまる時に実行されるプログラム」として認識します。つまり、**インデントされていない箇所＝文頭にスペースのない行**は条件に当てはまっていてもいなくても必ず実行されます。

テキスト

AI によって生成されたコンテンツは間違っている可能性があります。 テキスト

AI によって生成されたコンテンツは間違っている可能性があります。

図4.4 if文内のプログラムとif文外のプログラム

　条件文には、表4.1のような種類があります。=（イコール1つ）は代入を表し、==（イコール2つ）は等式を表すため、使い分けに注意が必要です。

表2.1 使用できる条件文

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 種類 | 記号 | 機能 | 使用例 | 使用できる型 |
| 等式 | == | 左辺と右辺が等しい | a == b | 数値型・文字列型 |
| 不等式 | != | 左辺と右辺が等しくない | a != b | 数値型・文字列型 |
| 大なり | > | 左辺の方が右辺より大きい | a > b | 数値型 |
| 大なりイコール | >= | 左辺が右辺以上の値である | a >= b | 数値型 |
| 小なり | < | 左辺の方が右辺より小さい | a < b | 数値型 |
| 小なりイコール | <= | 左辺が右辺以下の値である | a <= b | 数値型 |

　if文には複数行のプログラムを入れることもできます。下はaが9より小さい時のみ、aに10を加えてaを出力するプログラムです。

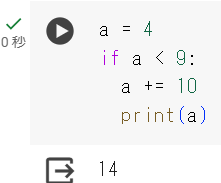


図4.5 if文内に複数行のプログラムを記述する

　演算子の左辺・右辺には計算式を入れることも可能です。

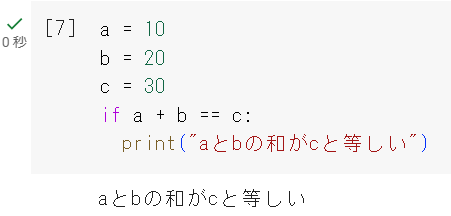


図4.6 計算式を条件文に含む

　偶数や奇数を判定する時は、図4.7のようなプログラムを作成します。偶数は、2で割った時の余りが0となります。

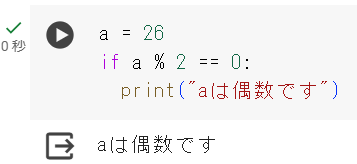


図4.7 偶数を判定するプログラム

**●練習問題4-1**

　今日の最高気温を調べ、変数temperatureに代入しましょう。そして、今日が夏日であったら「夏日です」と出力するプログラムを記述しましょう。夏日の定義を知らない場合は調べてみましょう。

**●解答例4-1**

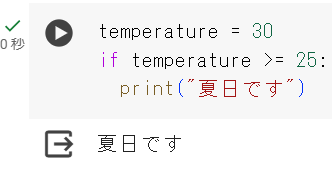


図4.8 解答例4-1

* 1. else文

　else文は、必ずif文と同時に使用されます。if文の条件に当てはまらない時にプログラムを実行します。else文を日本語で表現すると「違えば」となります。

　else文は図4.9のような形式で、if文の直後に記述を行います。「else:」と記述をした後に、インデントをしてプログラムを記述します。if文の条件に当てはまる時はプログラム①、条件に当てはまらない時はプログラム②が実行されます。

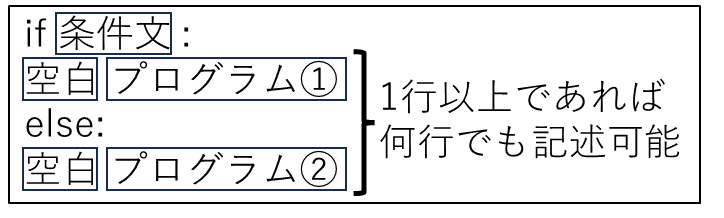


図4.9 else文の構造

　else文を用いると、図4.10のようなプログラムを記述することができます。

① 1行目で、aに3を代入します。

② if文の条件文が正しいかどうかを確かめます。a > 5は正しくないため、if文内のプログラムは実行されず、else文内のプログラムが実行されます。

1行目でaに代入される数値を変えてみて、条件が正しい時にどのようなプログラムが実行されるかを確かめてみましょう。

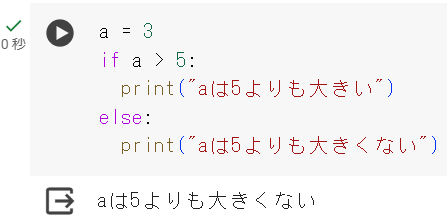


図4.10 else文を用いたプログラム

**●練習問題4-2**

　練習問題4-1で作成したプログラムにelse文を追加して、夏日でない時は「夏日ではありません」と出力するプログラムを記述しましょう。

**●解答例4-2**

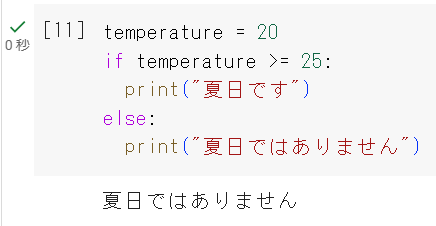


図4.11 解答例4-2

* 1. elif文

　elif文は、必ずif文と同時に使用されます。if文の条件が正しくない時に、elif文の条件が正しいかどうかを確かめます。その条件が正しければ、指定したプログラムを実行することができます。elif文を日本語で表現すると「違えば、もし～ならば」となります。

　elif文は、図4.12のような形式で記述します。図4.12のプログラムは次のように実行されます

① まず、条件文①が正しいかどうかを確かめます。正しければ、プログラム①が実行され、elif文より下のプログラムは全て実行されません。

② 条件文①が正しくない時、elif文の条件②が正しいかどうかを確かめます。正しければプログラム②が実行され、そのelif文より下のelif・else文は実行されません。

③ 手順②と同じ処理を条件文・プログラム③でも行います。

④ 条件文①②③が全て異なっている場合、else文内のプログラム④が実行されます。

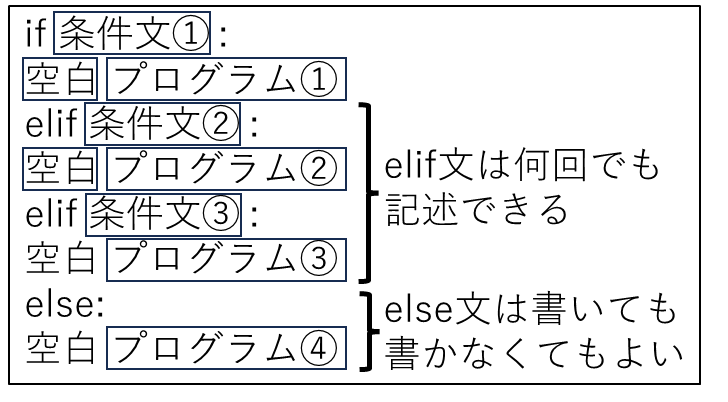


図4.12 elif文の構造

　if・elif文は上から正しいかどうかを確かめます。正しければ、if・elif文内のプログラムが実行されます。正しくなければ、if・elif文内のプログラムは実行されず、次のelif文の条件を確認します。以上を繰り返し、全て正しくない場合はelse文内のプログラムが実行されます。else文を記述しなかった場合は何も実行されません。

　図4.13は、elif文を用いたプログラムです。

① 変数aに4を代入します。

② 条件文a > 5が正しくないので、if文内のプログラムは実行されません。

③ 条件文a > 3が正しいため、elif文内のプログラムが実行されます。

④ elif文が正しかったため、else文内のプログラムは実行されません。

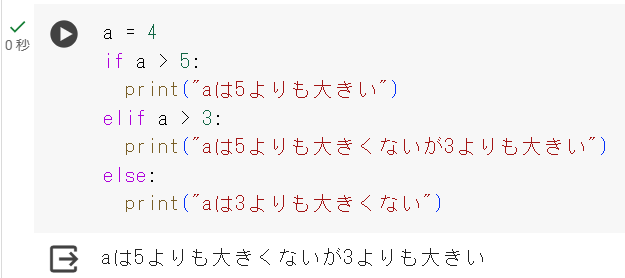


図4.13 elif文を用いたプログラム

また、if文が正しかった場合は、elif文が正しくてもelif文内のプログラムは実行されません。図4.14はプログラムの例です。

① 変数aに6を代入します。

② 条件文a > 5が正しいため、if文内のプログラムが実行されます。

③ if文が正しかったため、elif・else文内のプログラムは実行されません。

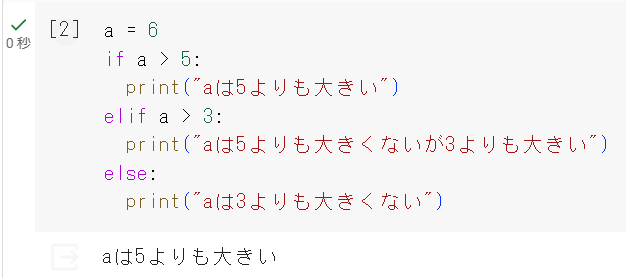


図4.14 条件に当てはまっていても実行されないelif文

　elif文は、複数記述することができます。図4.15はプログラムの例です。

① 変数aに7を代入します。

② 条件文a >= 10が正しくないため、if文内のプログラムは実行されません。

③ 条件文 a >= 8が正しくないため、elif文内のプログラムが実行されません。

④ 条件文a >= 6が正しいため、elif文内のプログラムが実行されます。

⑤ それより以下のelif・else文は実行されません。

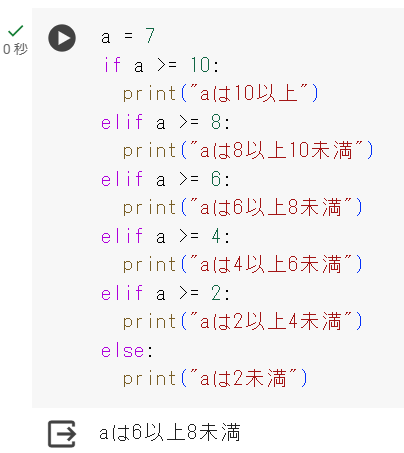


図4.15 複数のelif文

**●練習問題4-3**

　あなたは、テストの採点を行います。60点以上が合格、60点未満は不合格です。また、合格の場合はS・A・B・Cのどれかの成績が付きます。成績は、90点以上がS、80点以上90点未満がA、70点以上80点未満がB、60点以上70点未満がCです。変数scoreにテストの点数である0～100までの成績を代入し、合格ならば成績を、不合格ならば「不合格」と出力するプログラムを記述しましょう。

**●解答例4-3**

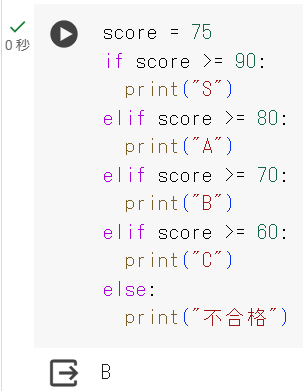


図4.16 解答例4-3

* 1. if文の入れ子

　if文の中にif文を記述することも可能です。そのような構造を**入れ子**といいます。入れ子は、図4.17のような構造のことを指します。外側と内側のif・else文を分けて考えます。

① 条件文①が正しいかどうかを確認します。

② 条件文①が正しい場合は、条件文②を確認します。条件文②が正しければプログラム①を、正しくなければプログラム②を実行します。

③ 条件文①が正しくない場合は、条件文③を確認します。条件文③が正しければプログラム③、正しくなければプログラム④を実行します。

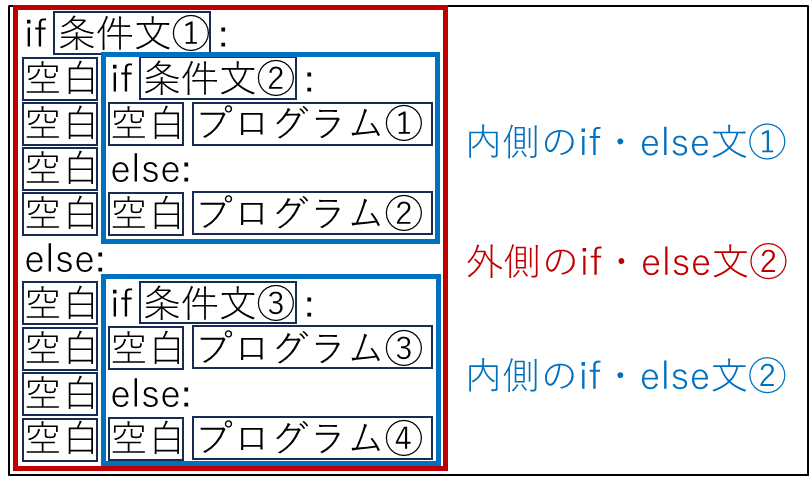


図4.17 elif文の構造

　if文の入れ子を用いると、図4.18のようなプログラムを記述することができます。

① 変数aに1を、変数bに3を代入します。

② if文の条件文a == 1が正しいかどうかを確かめます。変数aは1で正しいため、if文内のプログラムが実行されます。

③ if文の条件文b == 1が正しいかどうかを確かめます。変数bは3で正しくないため、if文内ではなくelse文内のプログラムが実行されます。

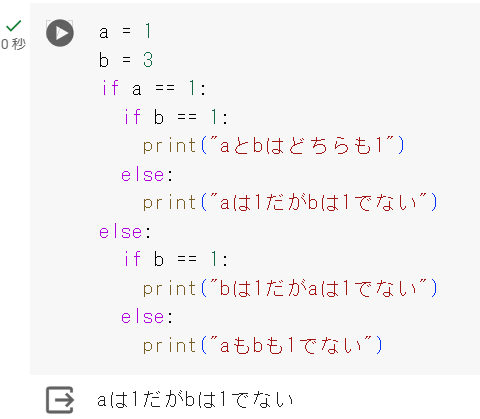


図4.18 入れ子を用いたプログラム

**●練習問題4-4**

　あなたは、AさんとBさんのテストの採点をしています。60点以上で合格、60点未満で不合格です。不合格の場合、再試験を受ける必要があります。変数score\_a・score\_bにAさん・Bさんの点数を0～100で代入します。そして、再試験を受ける必要がある人を出力しましょう。

・どちらも不合格の場合は「AさんもBさんも再試験を受ける」

・Aさんのみ不合格の場合は「Aさんのみ再試験を受ける」

・Bさんのみ不合格の場合は「Bさんのみ再試験を受ける」

・どちらも合格の場合は「AさんもBさんも再試験を受けない」

**●解答例4-4**

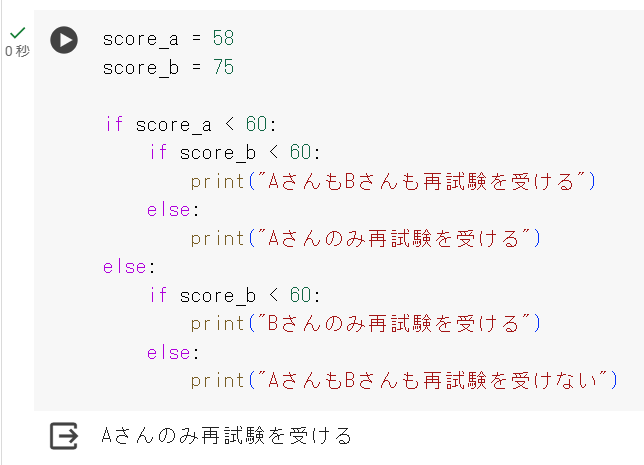


図4.19 解答例4-4

1. 配列
   1. 配列とは

　配列を用いると、値を連続的に保存することができます。図5.1は配列のイメージです。配列では、連続的に保存した値に番号を付けて管理します。配列では先頭にある要素は、1番目の要素ではなく0番目の要素となるので、注意が必要です。先頭の0番目から順番に1番目、2番目…となります。配列の値には、文字列・数値ともに使用できます。

要素　　　　 要素　　　 要素

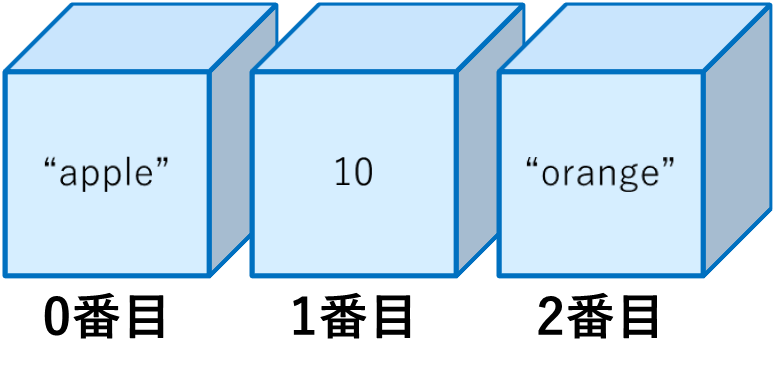


図5.1 配列のイメージ（a=[”apple”, 10, ”orange”]）

　配列を宣言する時は[　]（括弧）を用います。また、要素と要素は「,」（カンマ）で区切ります。

**a = [1番目の要素, 2番目の要素, 3番目の要素,…]**

のような形式で宣言をします。要素の個数は何個でもよいです。配列のn番目を指定する時は、配列[n]とします。

図5.2は配列を用いたプログラムです。

① 変数aに配列[“apple”,”banana”,”orange”]を代入します。この配列は、0番目の要素が”apple”、1番目の要素が”banana”、2番目の要素が”orange”であることを指します。

② 変数aを出力します。変数aには配列が代入されているため、配列が出力されます。

③ a[1]を出力します。a[1]は変数aの1番目という意味です。変数aには配列が代入されており、1番目の要素である”banana”が出力されます。

　0番目 　1番目　 2番目

　 a[0] a[1] a[2]

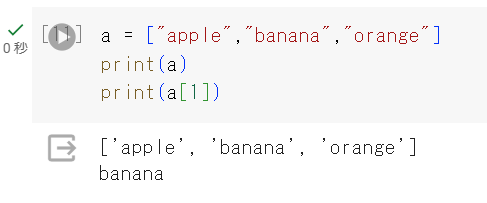


図5.2 配列を用いたプログラム

　図5.3のように、同一の配列に数値型と文字列型の両方を含むことも可能です。

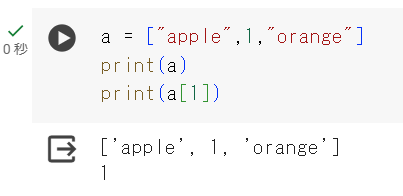


図5.3 数値と文字列が混在した配列

　配列の要素の値を変更する時は、変数の代入と同じような記述をします。配列のn番目の要素を変更する時は、

**配列[n] = 変更後の値**

とします。図5.4のプログラムでは、2行目で配列の値を変更しています。

① 変数aに配列[“apple”,”banana”,”orange”]を代入します。

② 1番目の要素に”pineapple”を代入します。変数aの1番目の値が”banana”から”pineapple”に変更されます。

③ 変数aを出力します。

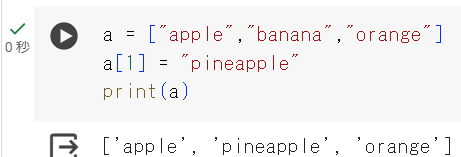


図5.4 配列の値の変更

* 1. 要素の追加

　配列に要素を追加することも可能です。配列の末尾に要素を追加する時はappendを使用します。

**配列.append(値)**

と記述すると、appendで指定した値で要素が追加されます。

　図5.5はappendを用いて要素を追加したプログラムです。

① 変数aに配列[20,3,4,5]を代入します。

② 変数aを出力します。

③ 変数a の末尾に10を追加します。

④ 変数aを出力します。

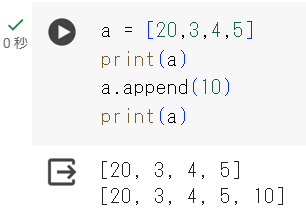


図5.5 配列に要素(10の値)を１つ追加した例

* 1. 値の削除

　配列の要素を削除することもできます。配列の{数値}番目の要素を削除する時はpopを使用します。

**配列.pop(数値)**

　図5.6のプログラムは、popを使用して要素を削除したプログラムです。

① 変数aに配列[“egg”,”chicken”,”tomato”]を代入します。

② 変数aを出力します。

③ 変数aの1番目の要素、つまり”chicken”を削除します。

④ 変数aを出力します。

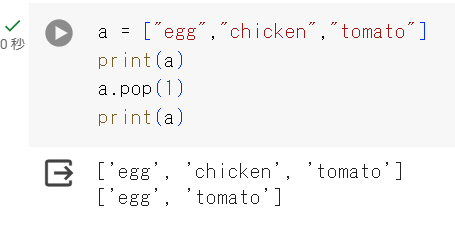


図5.6 要素を１つ削除する

* 1. 配列の長さを調べる

　配列の長さを取得する時には、len関数を使用します。

**len(配列)**

とすることで、配列の長さを取得できます。図5.7はlen関数を用いたプログラムです。

① 変数aに配列[“egg”,”chicken”,”tomato”]を代入します。

② 変数aとその長さを出力します。

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション, チャットまたはテキスト メッセージ

自動的に生成された説明

図5.7　aの配列の長さ（個数）を調べる

③ 変数aの末尾に”beaf”を追加します。

④ 変数aとその長さを出力します（beafを追加したので、数が１つ増えているはずです）。

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション, チャットまたはテキスト メッセージ

自動的に生成された説明

図5.8　aの配列の長さ（個数）を調べる

1. ループ

　同じプログラムを複数回実行することをループと呼びます。ループは、for文を使ったものとwhile文を使ったものの2種類が存在します。

* 1. for文

　for文は、指定した回数同じプログラムを実行することができます。図6.1はfor文の構造です。range関数の中にある「回数」に数値を入力すると、その回数プログラムが実行されます。

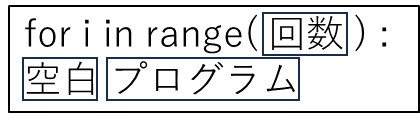


図6.1 for文の構造

　for文を用いると、図6.2のようなプログラムを記述することが可能です。このプログラムでは、回数の部分に3が入力されているため、print(“Hello World”)が3回実行されます。

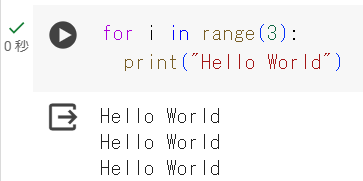


図6.2. for文を用いたプログラム

　for i inのiは変数です。iがよく使用されますが、他の変数名を使用することも可能です。変数の中身を調べるために、print関数でiを出力してみましょう。

　図6.3はiを出力するプログラムです。iには、指定した回数まで、数値が順に代入されます。

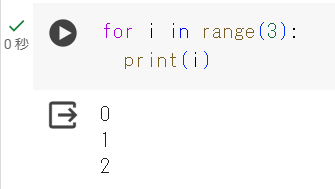


図6.3 for文のiを出力する

* 1. 配列の各要素の参照

　for文を用いて、配列の要素を1つずつ参照することが可能です。その場合、図6.4のような構造になります。変数iに、配列の0番目から順に要素が代入されます。

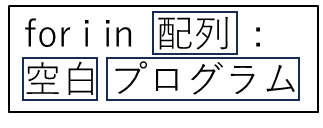


図6.4 配列の要素を1つずつ参照する

　図6.5はfor文を用いて、配列の各要素を順に取得するプログラムです。iに”egg”,”chicken”,”tomato”の順に代入されます。

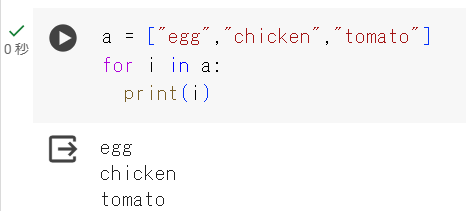


図6.5 配列の各要素の参照

* 1. while文

　while文は、特定の条件に当てはまっている時に同じプログラムを繰り返し実行します。while文は図6.6のような構造で記述します。

① 条件式が正しいかどうかを確かめます。正しくない場合はプログラムを実行しません。

② 正しい場合は、プログラムを実行します。プログラムを実行し終えたら、再度①に戻ります。

このように、条件が正しい時はずっとwhile文内のプログラムを繰り返し実行します。



図6.6 while文の構造

　while文を用いると、図6.7のようなプログラムを記述することができます。

① 変数aに0が代入されます。

② while文の条件式a < 3が正しいため、while文内のプログラムが実行されます。

③ 変数aが出力されます。そして、変数aに1が加算され、変数aは1になります。

④ while文の条件に戻ります。条件式a < 3が正しいため、while文内のプログラムが実行されます。

⑤ 変数aが出力されます。そして、変数aに1が加算され、変数aは2になります。

⑥ while文の条件に戻ります。条件式a < 3が正しいため、while文内のプログラムが実行されます。

⑦ 変数aが出力されます。そして、変数aに1が加算され、変数aは3になります。

⑧ while文の条件に戻ります。条件式a < 3にあてはまらないため、while文内のプログラムが実行されません。

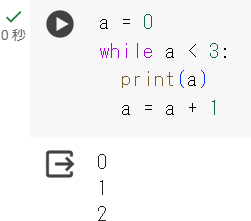


図6.7 while文を用いたプログラム

　while文を用いると、for文と同じように指定した回数繰り返すプログラムを作成することができます。図6.8は指定した回数をn回繰り返すプログラムです。変数aに0からn-1までの値が順に加算されるため、n回繰り返されます。

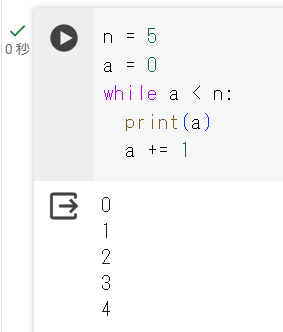


図6.8 while文を用いたn回繰り返すプログラム

　while文で、配列を0番目から順に参照することも可能です。図6.9は、配列の番号の付け方のイメージ図です。最初の要素が0番目、配列の長さはlen(配列)で表されるため、最後の要素はlen(配列)-1番目となります。

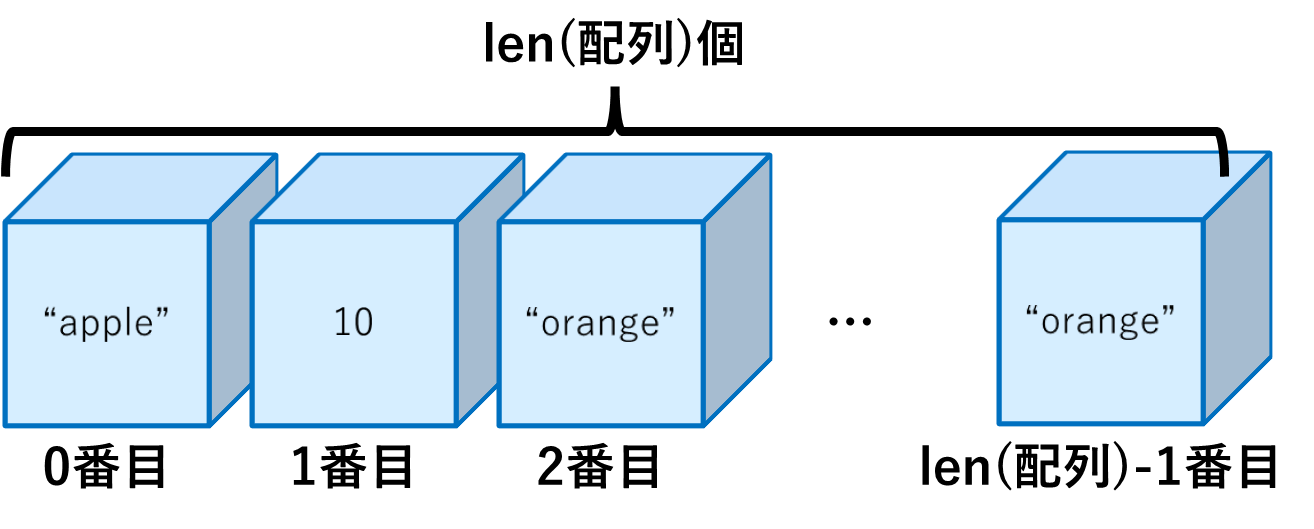


図6.9 配列の番号

　よって、0番目からlen(配列)-1番目までを繰り返すwhile文を記述すればよいことが分かります。図6.8のプログラムを元に作成すると、図6.10のようになります。

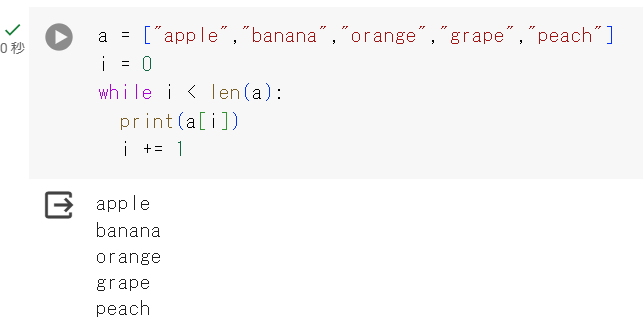


図6.10 while文を用いて配列の要素を順に参照する

同じ考えをfor文でも行うことができます。図6.11のプログラムのfor文では、変数iに0からlen(a)-1までの数値が順に代入されるため、配列の各要素を順に参照することができます。



図6.11 for文を用いて配列の要素を順に参照する

**●練習問題6**

　数値だけの配列を作成し、変数numbersに代入しましょう。そして、numbersを全て2倍するプログラムを作成し、numbersを出力しましょう。

**●解答例6**

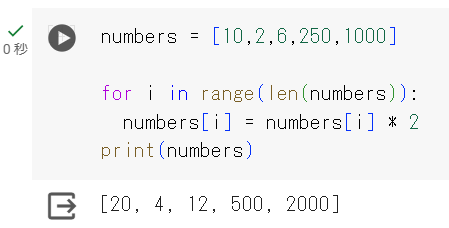


図6.12 解答例6

1. 関数
   1. 関数とは

　一連の処理をひとまとまりにすることを関数と言います。関数を用いると、特定の処理を何度も実行することができます。関数を定義することを**宣言**、定義した関数を呼び出すことを**呼び出し**といいます。

　関数は、図7.1のような構造で宣言・呼び出しが行われます。defを用いて宣言が行われます。

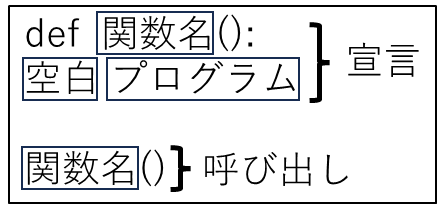


図7.1 関数の宣言・呼び出し

　図7.2は、関数を用いたプログラムです。

① print\_hello関数を宣言します。print\_hello関数は、”Hello!”と出力するプログラムです。

② print\_hello関数を呼び出します。print\_hello関数に含まれる、”Hello!”と出力するプログラムが実行されます。

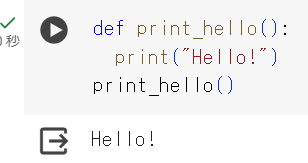


図7.2 関数を用いたプログラム

　一度宣言された関数は、何度でも呼び出すことができます。

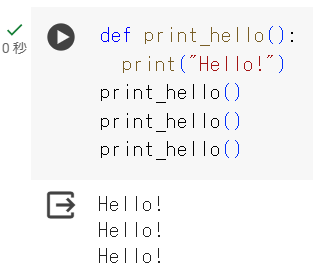


図7.3 関数を複数回呼び出すプログラム

* 1. 関数の引数

　関数には引数を設定することができます。引数を用いると、関数を実行する時に値を渡すことができます。図7.4は引数を設定した関数の構文です。「引数1、引数2、…」はそれぞれ「値1、値2、…」と1対1で対応します。引数は変数のように扱われ、値1が引数1に、値2が引数2に代入されます。引数の数は1つ以上であれば何個でも設定可能です。引数を設定しないことも可能です。

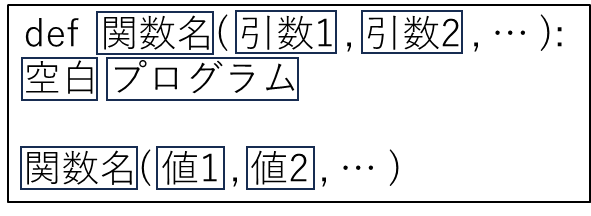


図7.4 引数を設定した関数の構文

　図7.5は関数の引数を用いたプログラムです。

① print\_sentence関数が宣言されます。引数1にsentenceが設定されています。print\_sentence関数は、引数の1のsentenceを出力するプログラムです。

② print\_sentence関数が呼び出されます。値1に”Hello!”が指定されています。print\_sentence関数の引数1はsentenceであるため、sentenceに”Hello!”が代入された状態でprint\_sentence関数のプログラムが実行されます。

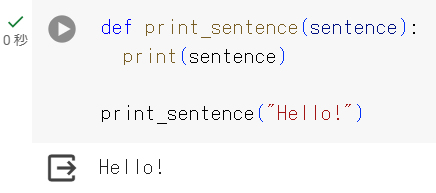


図7.5 引数を設定した関数を含むプログラム

　図7.6は、引数を複数設定した場合です。

① plus関数が宣言されています。引数1がa、引数2がb、引数3がcです。plus関数は、aとbとcの和を出力するプログラムです。

② 値1に5、値2に8、値3に9を指定した状態でplus関数を呼び出します。aには5、bには8、cには9が代入された状態で、plus関数が実行されます。

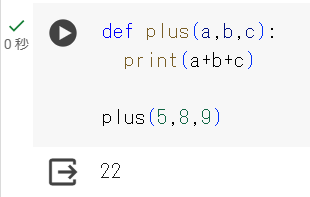


図7.6 引数を複数設定した関数を含むプログラム

* 1. ローカル変数とグローバル変数

　関数内と関数外で宣言された変数は扱い方が異なります。関数内で宣言された変数をローカル変数、関数外で宣言された変数をグローバル変数と言います。ローカル変数は関数外で使用することはできません。

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション, メール

AI によって生成されたコンテンツは間違っている可能性があります。

図7.7 関数内で宣言された変数を関数外で使用するとエラーが出る

　また、グローバル変数と同じ名前の変数が関数内で宣言された時は、別の変数として扱います。

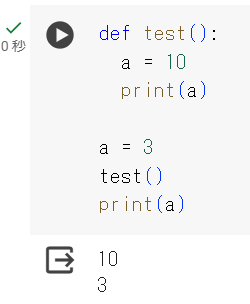


図7.8 ローカル変数とグローバル変数は異なる変数として処理される

* 1. 関数の戻り値

　戻り値を設定すると、関数で計算した結果を呼び出し元に返すことが可能です。戻り値を設定する時は、return文を用います。「return 値」とすることで、呼び出し元に値を戻します。

　図7.9のように、呼び出し元のプログラムが、returnの次に指定した値になるような構造となっています。

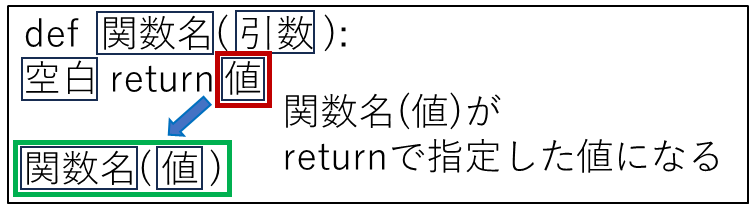


図7.9 関数の戻り値の構造

　図7.10は関数の戻り値を使用したプログラムです。

① plus関数を宣言します。引数1がa、引数2がb、引数3がcです。aとbとcを足して、その値を戻り値として指定します。

② 値1に2、値2に3、値3に10を指定した状態でplus関数を呼び出して、それを出力します。plus関数の引数のaに2、bに3、cに10が代入され、全て足してそれが戻り値になります。呼び出し元、つまりplus(2,3,10)が戻り値の値になります。その値が出力されるため、出力結果は15となります。

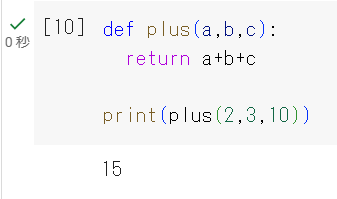


図7.10 関数の戻り値を使用したプログラム

　関数に条件分岐やループ文を含めることも可能です。図7.11は関数に条件分岐を含んだプログラムです。

① 引数1にaが設定されている関数even\_oddを宣言します。aを2で割った余りが0の時に”偶数”、そうでなければ”奇数”を戻り値にするプログラムです。

② 値1に10を指定した状態でeven\_odd関数を呼び出して、それを出力します。引数のaに10を代入します。aを2で割った時の余りが0であるため、”偶数”が戻り値となります。even\_odd(10)が戻り値の値になり、その値が出力されるため”偶数”と出力されます。

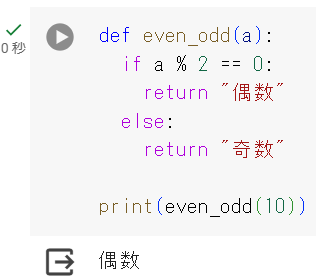


図7.11 関数に条件分岐を含んだプログラム

**●練習問題7**

　引数1をa、引数2をbとするdivCheck関数を作成します。divCheck関数はaをbで割りきれる時に”割り切れます”、割り切れない時は”割り切れません”と返します。

**●解答例7**

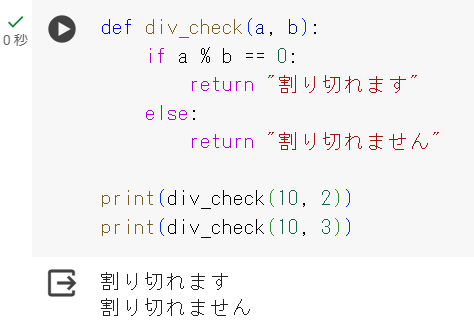


図7.12 関数に条件分岐を含んだプログラム

1. グラフィックス
   1. Turtle Graphicsのインストール

　Turtle Graphicsを使用すると、Pythonで図形を描くことができます。Turtle Graphicsを使用するためには、コードセルを追加して以下のコマンドを実行する必要があります。

!pip3 install ColabTurtle

コマンドを実行すると、以下のような表記がされます。

Successfully installed ColabTurtle-2.1.0

これでTurtle Graphicsがインストールされ使用できるようになります。ColabTurtle-2.1.0の2.1.0はバージョンを表しているので、他の数値になっていても問題ありません。



図8.1 コマンドの実行画面

　インストールが完了したら、Turtle Graphicsを使用できるようにします。再度コードセルを追加して、以下のプログラムを実行しましょう。

from ColabTurtle.Turtle import \*



図8.2 Turtle Graphicsを使用できるようにする。

　これで、Turtle Graphicsが使用できるようになります。以上の操作は、Turtle Graphicsを使用するために毎回必要な初期設定となります。Google Colaboratoryを閉じたり、時間が空いて再度使用したりする時には実行をした後にTurtle Graphicsを使用しましょう。

* 1. 基本的な操作

**・initializeTurtle()**

　Turtle Graphicsを実行する前に必要なプログラムです。このプログラムを実行すると、Turtle Graphicsの初期化がされ、画面が表示されます。画面中央に表示されている亀を移動させることによって図形を描画していきます。



図8.3 Turtle Graphicsの画面

・**forward()、backward()**

　亀を前や後ろに進ませるプログラムです。forward(数値)とすると、亀を数値分だけ前に進めます。backward(数値)とすると、亀を数値分だけ後ろに進めます。実際にforward()、backward()を使用して、亀を動かしてみましょう。移動した場所には線が引かれています。



図8.4 forward関数を使用して亀を前に進ませる

**・right()、left()**

　亀をその場で右や左に回転させるプログラムです。right(数値)とすることで、亀が{数値}度右に回転します。left(数値)とすることで、亀が{数値}度左に回転します。実際に、right()、left()を使用して亀を回転させてみましょう。図8.5は、forwardで前に進んでから右に90度回転させ、その後に再度forwardで前に進むプログラムです。

アプリケーション

低い精度で

図8.5 right関数を用いて亀を右に回転させる

**・goto(x,y)**

　亀を指定した場所に移動させるプログラムです。Turtle Graphicsは座標を用いて亀の位置を管理しています。左上が原点となります。x軸は右に行くほど値が大きくなり、0～800の範囲で指定できます。y軸は下に行くほど値が大きくなり、0～500の値の範囲で指定できます。初期の亀の位置は、(400,250)に位置しています。



図8.6 Turtle Graphicsの座標

　gotoのxにx座標の値、yにy座標の値を指定することによって、亀を指定した位置に移動させることができます。goto(100,200)とするとx座標が100、y座標が200の位置に亀を移動させます。図8.7は、(100,200)に移動した後に(700,300)に移動するプログラムです。

図形 が含まれている画像

自動的に生成された説明

図8.7 goto関数を使って亀を特定の表に移動させる

**・color()**

　color関数を用いると、これから引く線の色を変更することができます。color(色)とすることで、指定した色に変更することができます。実際に、color関数を使用して色を変更してみましょう。図8.8では、赤色で線を引き、その後色を青色に変更して再度線を引くプログラムです。

グラフィカル ユーザー インターフェイス

低い精度で自動的に生成された説明

図8.8 color関数を使用して線の色を変更する

　color関数で指定できる色は、数多く存在します。本教材では、表8.1に記載した色のみ使用します。使用できる色の種類は、colors manual page - Tk Built-In Commands（<https://www.tcl-lang.org/man/tcl8.6/TkCmd/colors.htm>）から取得することができます。

表8.1 本教材で使用する色

|  |  |  |
| --- | --- | --- |
|  | 色 |  |
|  | black |  |
|  | white |  |
|  | red |  |
|  | blue |  |
|  | yellow |  |

**・pendown()、penup()**

　亀は線を描かないで、移動することもできます。penup関数を実行すると線を描かずに亀が移動します。再び線を描きたいときは、pendown関数を実行すると、亀が移動するときに線を描くようになります。図8.9はpenup関数を使用して線を描かないようにしたのちに、pendown関数を使用して再び線を描くようにしたプログラムです。

グラフィカル ユーザー インターフェイス

自動的に生成された説明

図8.9 penup、pendown関数を用いて線を描くかどうかを設定する

　以上が、Turtle Graphicsで使用する関数です。

表8.2　Turtle Graphicsで使用する関数一覧

|  |  |
| --- | --- |
| **関数** | **機能** |
| initializeTurtle() | 画面の表示。TurtleGraphicsを使用する際、最初に記述する |
| forward(数値) | 亀を前に（数値）分動かす |
| backward(数値) | 亀を後ろに（数値）分動かす |
| right(数値) | 亀を右に（数値）度回転させる |
| left(数値) | 亀を左に（数値）度回転させる |
| pendown() | 亀が移動する時に線を引く |
| penup() | 亀が移動する時に線を引かない |
| color(色) | 引く線の色を変更する |

* 1. 図形を描いてみよう

　Turtle Graphicsの関数を駆使することで、図形を描くことができます。

**・正方形を描いてみよう**

　Turtle Graphicsを用いて、正方形を描いてみましょう。辺をforward関数で表し、各頂点でright関数（left関数）をもちいて亀を90度回転させることによって正方形を描くことができます。図8.10は、正方形を描くプログラムです。このプログラムでは、forward(100)で辺を表し、各頂点でright(90)を実行することで90度亀を回転させます。

グラフィカル ユーザー インターフェイス

低い精度で自動的に生成された説明

図8.10 正方形を描くプログラム

図8.10のプログラムでは、

forward(100)

right(90)

を4回繰り返しています。for文を用いることによって、プログラムを簡略化させること（より短い行数で同じプログラムを記述すること）ができます。

グラフィカル ユーザー インターフェイス が含まれている画像

自動的に生成された説明

図8.11 for文を用いて正方形を描くプログラム

　図8.11のプログラムを、関数を用いて呼び出せるようにしましょう。drawSquare関数を作成して、関数の中に正方形を描くプログラムを記述し、それを呼び出して正方形を描いてみましょう。

グラフィカル ユーザー インターフェイス

低い精度で自動的に生成された説明

図 8.12 関数を用いて正方形を描くプログラム

　図8.12のようにすると、drawSquare()と記述するだけで正方形を描くことができます。作成したdrawSquare関数を数回呼び出して、異なる場所に複数回正方形を描いてみましょう(図8.13参照)。

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

図8.13 複数の場所に正方形を描くプログラム

**・正三角形の描画**

　次は、正三角形を描画してみましょう。正方形の時と同じように、辺はforward関数、各頂点でright関数を用いて描画をしてみましょう。forwardで指定する値は100です。rightで指定する数値を考えてみましょう。正三角形の内角は60度ですが、亀が回転する角度は外角と同じ角度になります。外角は(180 - 内角)で計算することができるため、rightで指定する数値は120となります(図8.14参照)。

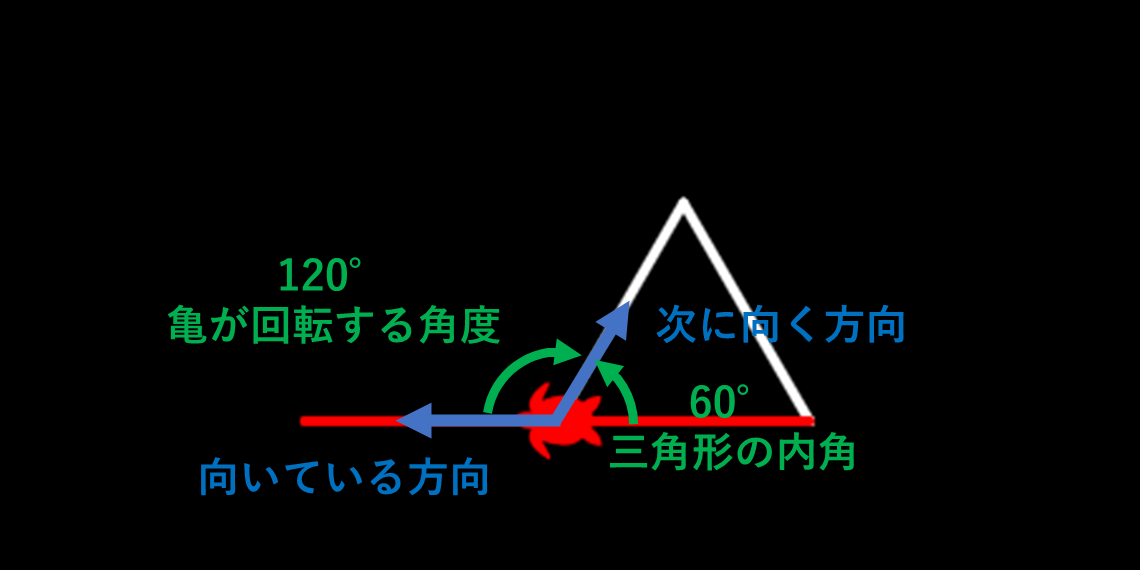


図8.14 亀が回転する角度の計算

　亀が回転する角度を計算出来たら、実際にプログラムを記述してみましょう。記述できたら、正方形の時と同じようにfor文を用いてプログラムを簡略化してみましょう。下記図8.15は正三角形を描くプログラム例です。

アイコン

自動的に生成された説明

図8.15 正三角形を描くプログラム

　そして、drawTriangle関数を作成して、正三角形を描くプログラムを呼び出せるようにしてみましょう（図8.16）。

グラフィカル ユーザー インターフェイス が含まれている画像

自動的に生成された説明

図8.16 関数を用いて正三角形を描くプログラム

　正三角形の大きさは、辺の長さによって変わります。辺が長いほど、正三角形の大きさは大きくなります。図8.17のプログラム例では、forward関数で指定した値によって正三角形の大きさが決まります。drawTriangle関数に引数sizeを設定して、正三角形の大きさを自由に変えられるようにしましょう。forward(size)とすることで、引数sizeに応じて値を変更することができます。呼び出すときは、引数sizeに代入する値を指定して呼び出します。

矢印 が含まれている画像

自動的に生成された説明

図8.17 正三角形のサイズを指定するプログラム

　正方形・正三角形は以上のようにして描画できます。次は、円を描画してみましょう。

**・円の描画**

　Turtle Graphicsで円を描画してみましょう。円を描画する時は、三角関数を用います。図8.18は円を描画するプログラムです。center\_xとcenter\_yが円の中心のx座標とy座標、radiusが半径を示します。speed関数を用いると、亀の移動速度を上げることができます。speed関数は1～13までの整数を入力すると、移動速度を設定でき、13が最も速く、1が最も遅くなります。円の描画は時間がかかるため、speed(13)として移動速度を速くします。

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

図8.18円の描画をするプログラム

　円の中心座標を一度(0,0)として考えてみましょう。円上にある点は(radius \* cosθ, radius \* sinθ)で表すことができます。θの値を0～360までの整数にすることで、円のすべての点の座標を調べることができます。実際には1.5や100.12などθの値が小数点の場合も計算する必要がありますが、そのような数は無数にあり描画に時間がかかってしまうため、θが整数の時のみの描画となっています。

ダイアグラム が含まれている画像

自動的に生成された説明

図8.19 円上にある点の座標の計算方法

　図8.19では中心座標を(0,0)としましたが、中心座標を(center\_x, center\_y)にする必要があります。全ての点のx座標にcenter\_xを、y座標にcenter\_yを足すと実現します。

ダイアグラム

中程度の精度で自動的に生成された説明

図8.20 中心座標を移動する

　プログラムでは、θの代わりにiを使用します。また、Pythonでsinやcosの計算をする際には角度はラジアンとして指定する必要があります。一周を360°とする度数法に対して、ラジアンは一周を2πとして計算します。ラジアンにするためには、math.radians(角度)とします。そして、ラジアン化した値を変数rに代入します。また、cosの計算はmath.cos(ラジアン)、sinの計算はmath.sin(ラジアン)とします。

表8.3 円を描くプログラム

|  |  |
| --- | --- |
| initializeTurtle()  speed(13)  penup()  center\_x = 400  center\_y = 250  radius = 50  for i in range(361):  r = math.radians(i)  x = center\_x + radius \* math.cos(r)  y = center\_y + radius \* math.sin(r)  goto(x,y)  pendown()  penup() | Turtle Graphicsの初期設定・画面の表示  スピードの設定  線を引かない設定にする  中心のx座標の設置  中心のy座標の設定  半径の設定  変数iを0～360にしながら361回繰り返す  　変数rに、iをラジアンにした値を代入  　x座標を計算し、変数xに代入  　y座標を計算し、変数yに代入  　亀を(x,y)に移動する  　線を引く設定にする  線を引かない設定にする |

このようにすることで、円を描画することができます。radius、center\_x、center\_yを引数にして、drawCircle関数を作成しましょう。そして、引数を好きな値にして様々な円を描いてみましょう。

ダイアグラム

低い精度で自動的に生成された説明

図8.21 drawCircle関数を作成して様々な円を描画する

1. 幾何学模様

　Turtle Graphicsを使用して幾何学模様を描いてみましょう。

* 1. 円を使った幾何学模様

　円のプログラムは第9章で作成をしました。記述したdrawCircle関数を使用して幾何学模様を描いてみましょう。drawCircle関数のfor文ですが、360回繰り返すと繰り返しに時間がかかりすぎてしまうため、図9.1のように変更しましょう。今までのプログラムでは、0～360度まで1度間隔で描画しましたが、8度間隔で描画します。繰り返し回数を46回にして、iに8を掛けることによって実現します。iが0,1,2,…の時に、8×iは0,8,16,…となるため、8度ずつの点を計算することが可能です。

テキスト

自動的に生成された説明

図9.1 変更したdrawCircle関数

　今回は図9.2のような幾何学模様を描画します。

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

図9.2 作成するプログラムと描画する幾何学模様

　それぞれの円の中心は、特定の円の円周上にあります。図9.3の赤色の円周上に、白色の円の中心である黄色の点があります。

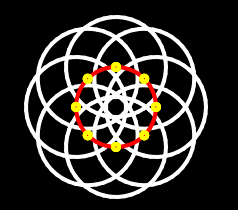


図9.3 それぞれの白色の円の中心は、赤色の円の上に存在

　赤色の円の中心を(0, 0)、半径をradiusとしたとき、それぞれの黄色の点の座標は

(radius \* cos0°, radius \* sin0°),(radius \* cos45°, radius \* sin45°) ,…

となります。for文を用いて、0～360度までの45度間隔の値を指定すればよいので、for i in range(8):とし、角度は45 \* iとすることで実現します。これは、for文を用いてiを0～7まで順に代入をして、角度の指定をするときにはiに45を掛けるプログラムです。赤色の中心座標は(center\_x, center\_y)とし、まずは画面の中心に表示させるので、center\_xに400、center\_yに250を代入しましょう。

表9.1 drawCircle関数を除いたプログラム

|  |  |
| --- | --- |
| initializeTurtle()  speed(13)  penup()  radius = 40  center\_x = 400  center\_y = 250  for i in range(8):  r = math.radians(i \* 45)  x = center\_x + radius \* math.cos(r)  y = center\_y + radius \* math.sin(r)  drawCircle(50,x,y) | Turtle Graphicsの初期設定・画面の表示  スピードの設定  線を引かない設定にする  半径の設定  中心のx座標の設置  中心のy座標の設定  変数iを0～7にしながら8回繰り返す  　変数rに、iをラジアンにした値を代入  　x座標を計算し、変数xに代入  　y座標を計算し、変数yに代入  　drawCircle(50,x,y)を実行 |

　幾何学模様を生成するプログラムを含むdrawGeometory関数を作成しましょう。drawGeometory関数には、radius・center\_x・center\_yを引数に設定しましょう。そうすると、drawGeometory関数を呼び出すだけで、幾何学模様が描かれます。何度も関数を呼び出して、綺麗な幾何学模様を生成してみよう。

グラフィカル ユーザー インターフェイス

自動的に生成された説明

図9.4 完全なプログラムと描いた幾何学模様

1. デジタル単語帳
   1. デジタル単語帳の設計

　デジタル単語帳は、配列を用いて単語の管理を行います。本教材では、日本語が表示されて英語を回答する単語帳を作成します。配列を用いて、単語とその回答を管理します。**変数wordsに出題される単語の配列**を、**変数answersに解答となる単語の配列**をそれぞれ代入します。

* 1. 入力を受け取る関数

　単語帳は、ユーザが回答を入力する必要があります。input関数を用いると、ユーザの入力した文字を受け取ることができます。input(ユーザに尋ねる文章)とすることで、文字の後にテキストを入力するフィールドが作成されます。input関数は、利用者が入力した文字を返す関数となっています。

　図10.1・図10.2のプログラムでは、input(“テキストを入力してください”)とすることで、「テキストを入力してください:」と出力されて、その後にテキスト入力フィールドが表示されます。入力した値がuser\_text変数に代入され、それを表示するプログラムとなっています。

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

図10.1 input関数のテキストフィールドが表示される

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

自動的に生成された説明

図10.2 入力した値を変数に代入して出力する

* 1. ランダムに値を代入してみよう

　Pythonには、ランダムに値を生成するrandomという機能があります。aとbを数値としたとき、random.randint(a,b)とすることで、a以上b以下のランダムな値を生成することができます。randomを使用する時は、使用する前のどこかの行にimport randomと入力する必要があります。

　random.randint(1,4)とすると、それぞれ25%の確率で1～4のどれかの整数が返されます。

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

図10.3 ランダムで値を生成する

* 1. 実際にプログラムを記述してみよう

　実際のデジタル単語帳のプログラムです。

表10.1 実際のプログラム

|  |  |
| --- | --- |
| import random  words = ["リンゴ","みかん","バナナ"]  answers = ["apple","orange","banana"]    while len(words) >= 1:      index = random.randint(0, len(words) - 1)      print(words[index] + "を英語に訳してください")      your\_answer = input("答え:")      if answers[index] == your\_answer:          print("正解!")          words.pop(index)          answers.pop(index)      else:          print("不正解...")  print("全問正解!") | ランダムモジュールのインポート  出題する日本語の単語の配列を変数wordsに代入  解答である英語の単語の配列を変数answersに代入  wordsの長さが1以上の時に繰り返す  　indexに0以上len(words)-1以下のランダム値の代入  　「words[index]を英語に訳してください」と出力  　your\_answerにユーザの入力を代入  　もし、answers[index]とyour\_answerが等しい時  　　「正解!」と出力  　　wordsのindex番目を削除  　　answersのindex番目を削除  　違えば  　　「不正解...」と出力  「全問正解!」と出力 |

　それぞれのプログラムは、次のようになっています。

1. randomをインポートして使えるようにする（1行目）
2. 単語の設定（2,3行目）

日本語と英語の単語セットを配列として宣言します。words配列0番目の日本語の単語には、answers配列0番目の英語の単語が、words配列1番目の日本語の単語には、answers配列1番目の英語の単語が…といったように、同じ番号同士の単語が対応します。

ダイアグラム

中程度の精度で自動的に生成された説明

図10.4 日本語と英語の単語を配列として宣言

1. ランダム値を生成して変数indexに代入（6行目）

変数indexに0以上len(words)-1以下のランダムな整数を代入します。wordsおよびanswers配列の最初の要素が0番目、最後の要素がlen(words)-1番目またはlen(answers)-1番目を指します。そのため、words配列とanswers配列から対応された単語を1組取得します。

図形 が含まれている画像

自動的に生成された説明

図10.5 index番目の要素を取り出す

1. ユーザに解答を尋ねる（7・8行目）

7行目のprint(words[index] + "を英語に訳してください")では、問題文を出力します。8行目のyour\_answer = input(“答え:”)では、ユーザに回答を尋ねて，ユーザの入力した文字を変数your\_answerに代入します。

1. ユーザの回答と正しい解答があっているかの確認（9行目）

　if文にて、answer[index]（正しい解答）とyour\_answer（ユーザが入力した回答）が等しいかどうかの確認をします。等しければユーザが入力した回答が正しい解答と一致しているので正解、そうでなければ不正解となります。

1. 正解の場合（10～12行目、answer[index]とyour\_answerが等しい）

正解の場合は、「正解！」と出力されます。そして、words・answers配列のindex番目を削除します。6～14行目は繰り返し行われるため、削除した要素が再び出題されることはありません。

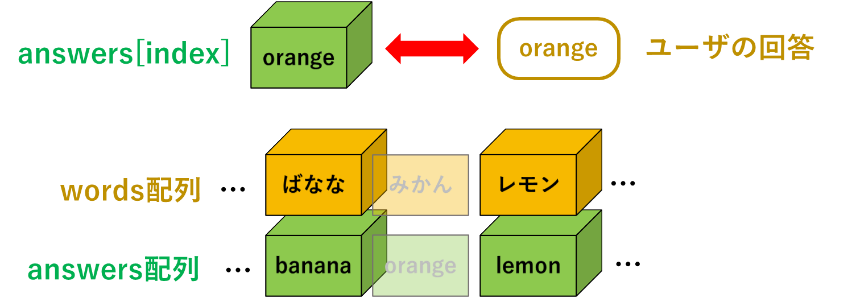


図10.6 正解した場合は要素を削除する

1. 不正解の場合（14行目、answer[indexとyour\_answerが等しくない）

不正解の場合は、「不正解！」と出力されます。配列の要素の削除が行われないため、不正解だった問題は再び出題されます。

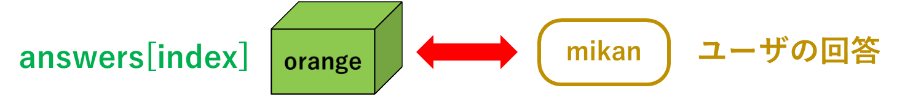


図10.7 不正解だった場合は要素を削除しない

1. words配列の長さが1以上の時にずっと繰り返す（5行目）

　正解をした時、words・answers配列が削除されるため、words配列に含まれる要素は未出題または不正解の問題となります。全ての問題に正解をしたら要素が空になります。Words配列が空になるまで繰り返されます。

ダイアグラム

AI によって生成されたコンテンツは間違っている可能性があります。

図10.8 　④〜⑧までの流れ：words配列が空になるまで繰り返す

　デジタル単語帳はこのような仕組み・プログラムとなっています。作成したプログラムを実行して、デジタル単語帳を使用してみましょう。不正解だった単語が再び出題されるかも確かめましょう。

テキスト

自動的に生成された説明

図10.9 単語帳プログラムの動作

　上手く動作したら、プログラムを改良してみましょう。例えば、配列に登録する単語を増やす、不正解の時に「不正解です。正解は○○です。」と出力するなど、使い勝手が良くなるような工夫をしてみましょう。

テキスト

自動的に生成された説明

図10.10 不正解の時に解答を提示してくれるプログラム

1. Hit and Blow
   1. 数値を文字列に変換する

　str関数を使うと数値を文字列に変換することができます。図11.1のプログラムでは、aを文字列に変換したものをbに代入します。aは数値なのでa+aは200が出力されます。bは文字列なのでb+bは100100が出力されます。

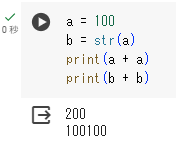


図11.1 数値を文字列に変換する

* 1. 文字列のn番目を取得する

　配列のn番目を取得する時は、配列[n]とします。それと同じく、文字列のn番目を取得する時は、文字列[n]とします。文字列も、先頭の文字は0番目となります。

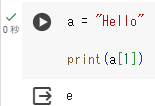


図11.2 文字列の1番目を取得

　文字列も、配列と同じでfor文で文字を参照することもできます。for i in 文字列:とすることで、文字列の先頭の文字から順にiに代入されます。

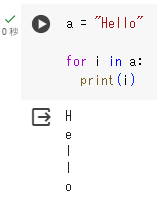


図11.3 文字列をfor文で参照

* 1. in演算子

　in演算子を用いると、文字列の中に特定の文字列が含まれるかを確認することができます。a in bとすると、bの文字列の中にaが入っているかどうかを確認できます。

テキスト

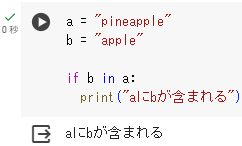
AI によって生成されたコンテンツは間違っている可能性があります。

図11.4 文字列に特定の文字列が含まれる場合（左）と配列に特定の要素が含まれる場合（右）

　図11.4の左では、pineappleの文字の中にappleが含まれるかを確認しています。右側では、aの配列の４つの要素の中に、6の数値が含まれているかを確認しています。

* 1. 条件式の「かつ」「または」

　andとorを用いると、条件式を組み合わせることができます。andを用いると、複数の条件式がどちらも正しい時にのみ実行するプログラムを作成することができます。orを用いると、複数の条件のどちらか正しい時に実行するプログラムを作成することができます。

　{条件式1} and {条件式2}とすると、条件式1と条件式2がどちらも正しい時に実行されます。{条件式1} or {条件式2}とすると、条件式1と条件式2の少なくともどちらか1つが正しい時に実行されます。orに関しては、どちらも正しい時も実行されます。

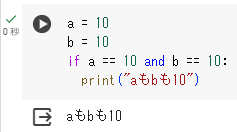


図11.5 andを用いた条件式

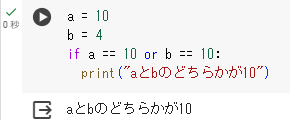


図11.6 orを用いた条件式

* 1. Hit and Blowのルール

　Hit and Blowは、3桁の数字を予想するゲームです。出題者と解答者に分かれます。

① 出題者は互いに数が重複しない3桁の数字を想像します。それが答えとなります。

② 解答者は、出題者の想像した数字を予測して、3桁の数字を言います。

③ 以下の規則に従って、出題者はHitとBlowの数の回答をします。

I. 予想した数値を1つずつ確認します。

II. 解答者の予想した1つの数が、答えと同じ位置にあった場合、Hitと数えます。

III. 解答者の予想した1つの数が、答えに含まれているが、位置が異なる場合はBlowと数えます。

IV. 「{ヒットの数}ヒット{ブローの数}ブロー」の形式で、回答者に伝えます。

④ 手順②③を繰り返し、3ヒットになる、つまり当たるまで繰り返します。

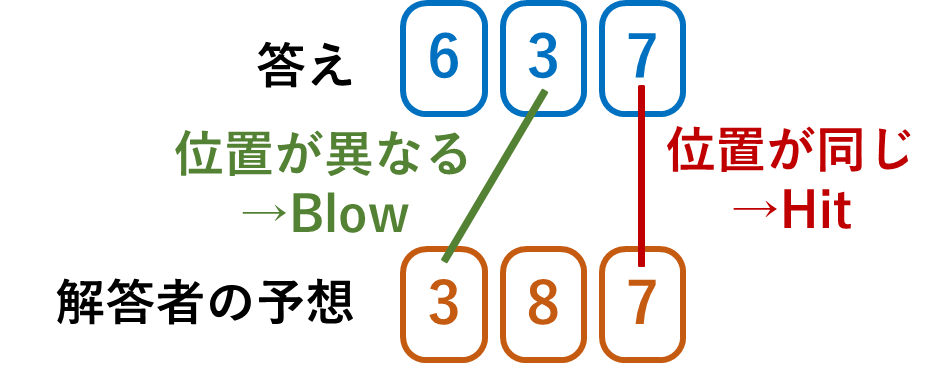
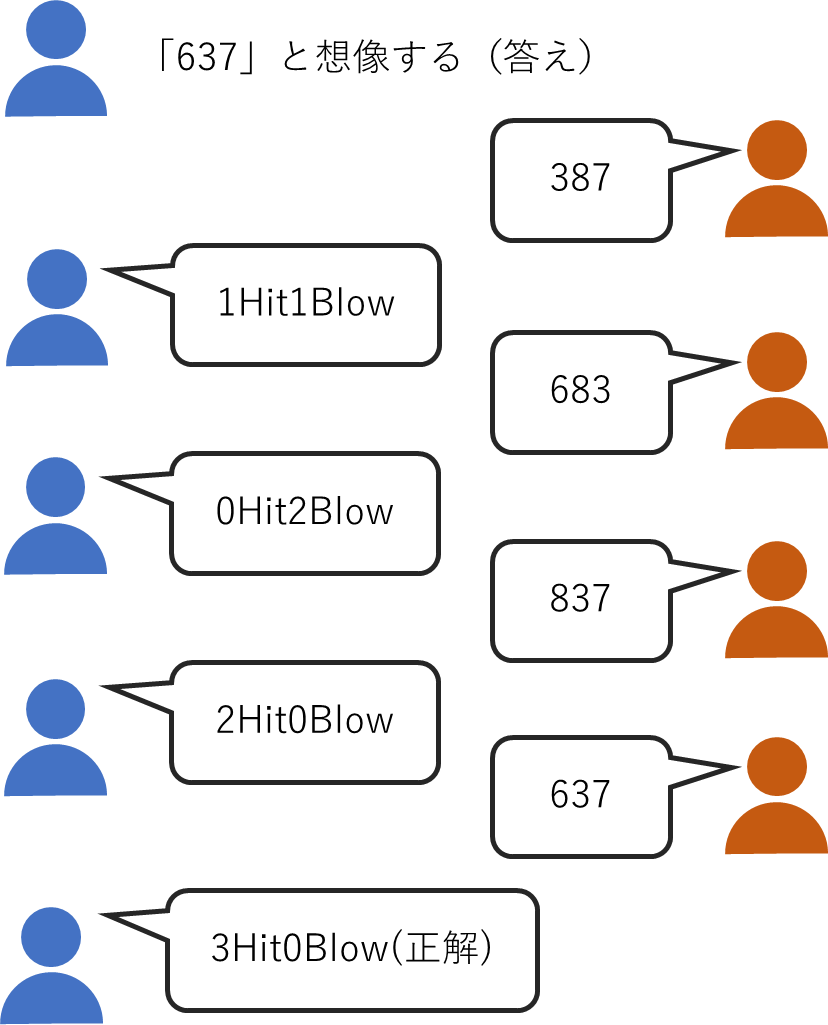


図11.7 HitとBlowのイメージ図



783

図11.8 やり取りの例

* 1. Hit and Blowをプログラムで記述する

　出題者のプログラムを組んで、ユーザに解答をしてもらいましょう。プログラムは、以下の通りです。

【プログラム例】

import random

numbers = []

for i in range(10):

  numbers.append(str(i))

select\_num = ""

for i in range(3):

  index = random.randint(0,len(numbers) - 1)

  select\_num = select\_num + numbers[index]

  numbers.pop(index)

user\_input = ""

count = 0

while select\_num != user\_input:

  user\_input = input("数値を予測してください:")

  hit = 0

  blow = 0

  for i in range(len(user\_input)):

    if user\_input[i] == select\_num[i]:

      hit += 1

    elif user\_input[i] in select\_num:

      blow += 1

  print(str(hit) + "Hit" + str(blow) + "Blow")

  count += 1

print(str(count) + "回で正解しました！")

① 0～9までの整数を配列としてnumbersに代入（3～5行目）

　空配列のnumbers配列を宣言します。for i in range(10)で、iに0～9までの整数を順に代入します。そして、iを文字列に変換した後にnumbersに追加します。下のプログラムと同義です。

numbers = [“0”,”1”,”2”,”3”,”4”,”5”,”6”,”7”,”8”,”9”]

② select\_numに互いに異なる3桁の数字を代入する（7～11行目）

　numbersから1つ選んでselect\_numに結合した後に、numbersから選んだ要素を削除するプログラムを3回繰り返します。9行目で、選択する要素をindexに代入します。10行目で、select\_numに選択した要素を結合します。11行目で、選択した要素をnumbersから削除します。

③ ユーザが数字を当てる（13～25行目）

　user\_inputはユーザの入力した文字を格納する変数、countはユーザが回答をした回数を格納する変数です。

15行目の、whileの条件文であるselect\_num != user\_inputでは、答え（select\_num）がユーザの答え（user\_input）と等しくないときに繰り返されます。つまり、正解をしたらループが終わります。

　16行目ではinput関数を呼び出して、ユーザの入力をuser\_input変数に代入します。17・18行目では、Hit・Blowの数を格納するhit・blow変数に0を代入します。

　19～23行目では、hitとblowの個数を数えます。for文で、user\_inputの0、1、2文字目を順に確認します。user\_input[i]がi番目の文字、select\_num[i]がi番目の文字です。user\_input[i]とselect\_num[i]が等しい時、同じ位置に同じ数字があるのでhitを1加算します。user\_input[i]がselect\_numに含まれている時は、同じ位置に同じ数字がなく正解にuser\_input[i]が含まれている、つまり、位置は異なるが同じ数字が含まれているのでblowを1加算します。

　24行目でHitとBlowの数を出力して、25行目でcountを1加算します。

④ 何回で正解したかを出力する（27行目）

　countには、何回回答したかが格納されています。countを出力することによって、回答回数を出力します。

* 1. もっと使いやすくしてみよう

　先ほど作成したプログラムでは、3桁の数字ではなくても入力ができたり、入力する3桁の数字が互いに異なっていなかったりしても入力を受け付けます。ユーザが誤って入力した時に、正しく入力するように教えるプログラムに改良しましょう。

① ユーザが入力した文字が3桁でないと受け付けないプログラム

　16・17行目の間に挿入します。文字列の長さはlen(文字列)で取得できます。それが3ではない時に、3桁で回答をすることを促せばよいので、下のプログラムを記述します。

  if len(user\_input) != 3:

    print("数字は3桁の必要があります。")

② 互いに異なっていない時に受け付けないプログラム

　①の次に挿入します。3桁の中に、同じ文字が存在することをif文の条件文で表します。0文字目と1文字目が同じ、または、1文字目と2文字目が同じ、または、0文字目と2文字目が同じであれば、同じ文字が存在することが示されます。プログラムで記述すると、次のようになります。①でif文を使用したので、elif文に記述します。

  elif user\_input[0] == user\_input[1] or user\_input[1] == user\_input[2] or user\_input[0] == user\_input[2]:

    print("数字は互いに異なる必要があります。")

③ ①②以外の場合は文字を受け取るプログラム

　①②はif・elif文として記述したので、hit = 0～count += 1までをelse文に記述します。完全なプログラムは以下になります。

【プログラム例】

import random

numbers = []

for i in range(10):

  numbers.append(str(i))

select\_num = ""

for i in range(4):

  index = random.randint(0,len(numbers) - 1)

  select\_num = select\_num + numbers[index]

  numbers.pop(index)

user\_input = ""

count = 0

while select\_num != user\_input:

  user\_input = input("数値を予測してください:")

  if len(user\_input) != 3:

    print("数字は3桁の必要があります。")

  elif user\_input[0] == user\_input[1] or user\_input[1] == user\_input[2] or user\_input[0] == user\_input[2]:

    print("数字は互いに異なる必要があります。")

  else:

    hit = 0

    blow = 0

    for i in range(len(user\_input)):

      if user\_input[i] == select\_num[i]:

        hit += 1

      elif user\_input[i] in select\_num:

        blow += 1

    print(str(hit) + "Hit" + str(blow) + "Blow")

    count += 1

print(str(count) + "回で正解しました！")

　プログラムを更に改良して、Hit and Blowを楽しくしてみましょう。

1. 迷路
   1. 二次元配列

　以下のように、配列の中に配列を入れることも可能です。変数名[数値1]で各配列の指定、変数名[数値1][数値2]で各要素の指定をすることができます。下の要素では、例えばa[1]は[4,5,6]を指します。また、a[0][2]は3を指します。

a = [[1,2,3],[4,5,6]]

　二次元配列を参照する時は、2重にfor文を使います。このコードは、iにaに含まれる配列を順に代入して、jにiの要素を順に代入して出力します。iには[1,2,3]、[4,5,6]が順に代入されます。

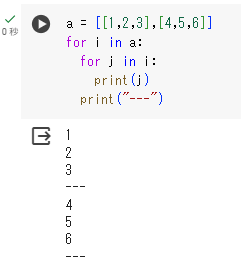


図12.1 二次元配列の出力

　range、len関数を用いて次のように記述することもできます。

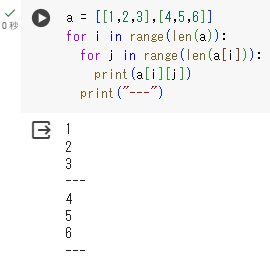


図12.2 range、len関数を用いた二次元配列の出力

* 1. 関数の再帰

　関数の中で、同じ関数を呼び出すことを再帰といいます。例えば、for文を用いてある数値nを用いて、0～nまでの数値の合計を求めるプログラムは次のようになります。

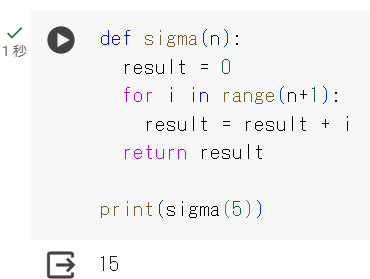


図12.3 0～nまでの数値の合計を求めるプログラム

　関数の再帰を用いると、次のようになります。

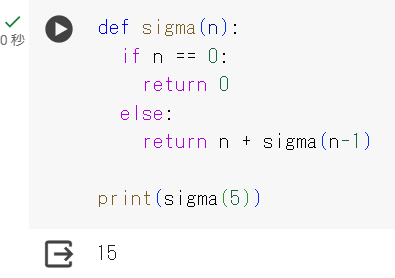


図12.4 関数の再帰を用いて0～nまでの合計を求めるプログラム

図12.5は関数の再帰のイメージ図です。各関数の上段が引数nの値、下段がreturnで返す値を示します。sigma(5)がsigma(4)を呼び出し、sigma(4)がsigma(3)を呼び出し、…、sigma(0)は0を返すといったように、sigma関数がsimga関数を呼び出しています。

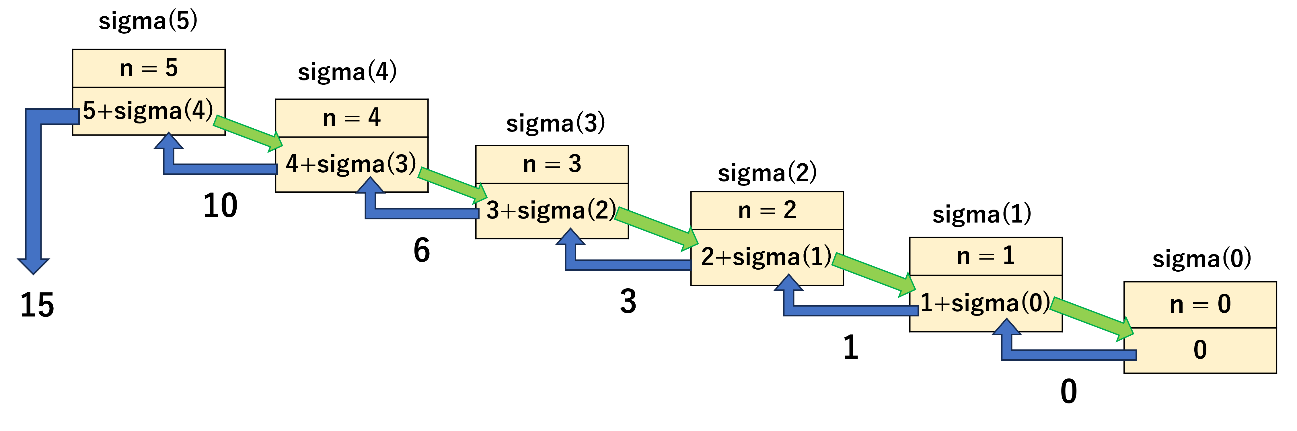


図12.5 関数の再帰のイメージ図

* 1. 迷路の生成アルゴリズム

　迷路を生成するアルゴリズムとして、棒倒し方があります。棒倒し方は、以下の手順で迷路が生成されます。

1. 奇数×奇数のマスを用意する

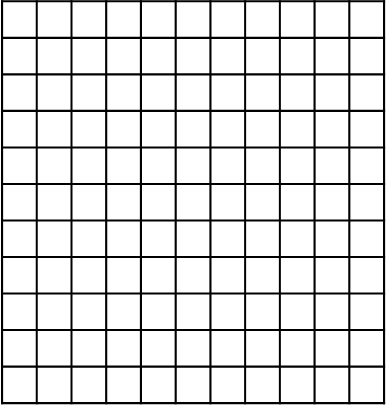


図12.6 迷路生成アルゴリズム①

1. 外周を壁にする

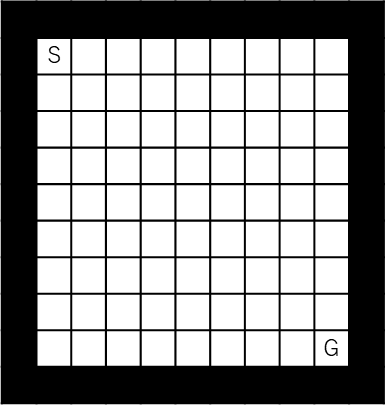


図12.7 迷路生成アルゴリズム②

1. 外周の内側に、1マスずつ壁を設置する

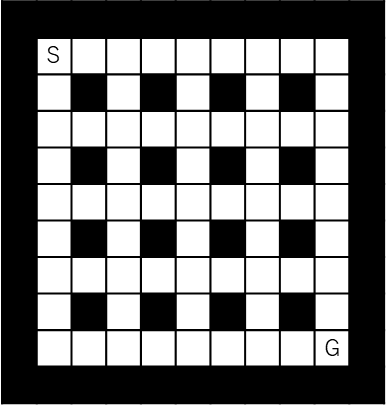


図12.8 迷路生成アルゴリズム③

1. ②で設置した壁を1つずつ順に全て選択し、以下の条件に当てはまる4方向の内、ランダムな方向に壁を設置する

i. 一番上の行の内側の壁のみ、上側に倒しても良い。

ii. すでに壁が設置されている場所には壁を設置しない。

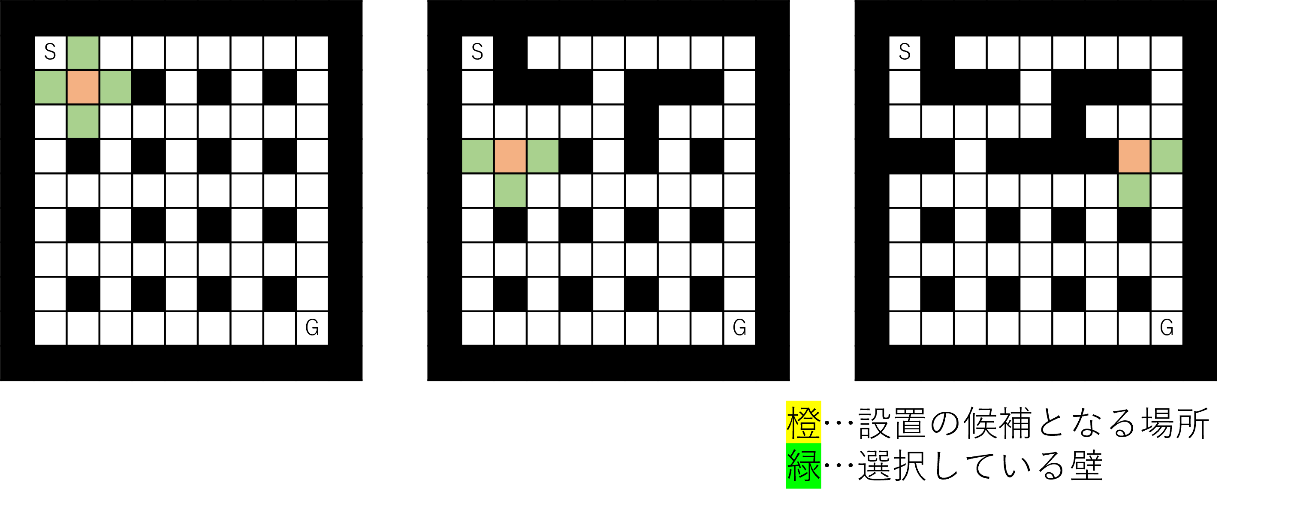


図12.9 迷路生成アルゴリズム④

　以上の処理をすることで、迷路を作成することができます。

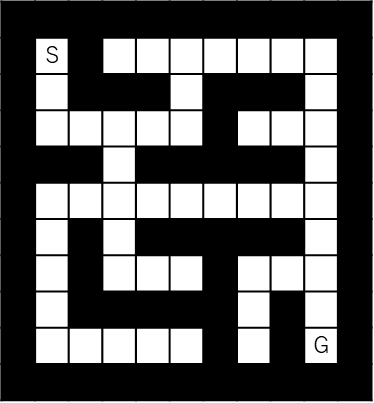


図12.10 迷路生成アルゴリズム⑤

* 1. 迷路の解法アルゴリズム

　迷路を解くアルゴリズムとして、深さ優先法があります。深さ優先探索は、以下の手順で迷路を解きます。

1. スタートから、順に探索をする。
2. 枝分かれ地点に着いた時は、とりあえずどこかの方向へ向かう。
3. 行き止まりに着いた時は、最後に通った枝分かれ地点まで戻り探索をしていない箇所へ探索します。全ての方向を探索した時は、その前に通過した枝分かれ地点に戻ります。

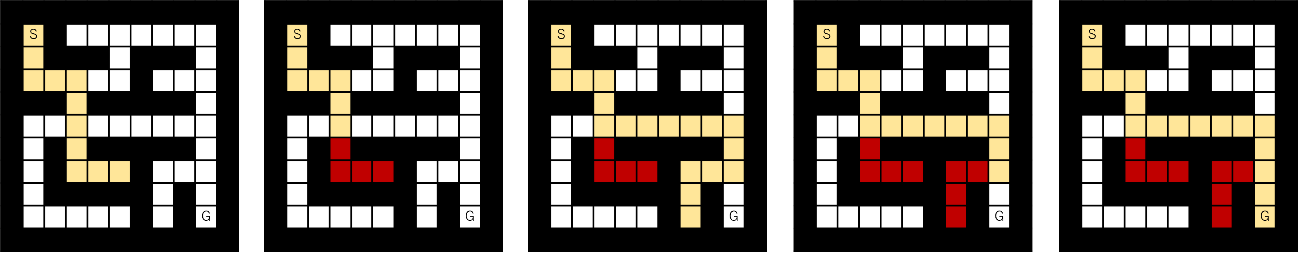


図12.11 迷路解法アルゴリズム

　迷路の生成・解法アルゴリズムをアニメーションにしたものを、<https://kyurururn.github.io/maze_sample/>から使えます。

* 1. 配列からランダムで選ぶ

　整数をランダムで生成することができるrandomですが、random.choice(配列)というように、配列からランダムで1つの要素を選択することもできます。

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

自動的に生成された説明

図12.12 配列からランダムに要素を選ぶ

* 1. 迷路の設計

　迷路は、二次元配列を変数mazeに代入して管理します。迷路の各タイルは次のように表します。

* 0 … 通り道
* 1 … 内側の壁
* 2 … 外側の壁
* 3 … スタート
* 4 … ゴール



図12.13 左から、迷路のイメージ図・数値変換・二次元配列

　また、迷路の縦はHEIGHT、迷路の横はWIDTHにそれぞれ代入します。

* 1. 迷路の生成プログラムの作成

　迷路の生成プログラムを作成しましょう。迷路の生成プログラムでは、grid()・create\_maze()・display\_maze()の3つの関数を用います。

* display\_maze() … 迷路を出力するプログラム
* grid() … 迷路の内側に1マスずつ壁を設置したものを生成するプログラム（12.3章 手順①～③）
* create\_maze() … 迷路の内側の壁をランダムに配置するプログラム（12.3章 手順④）

【プログラム例】

import random

def display\_maze(maze):

  for i in range(len(maze)):

    for j in range(len(maze[i])):

      if maze[i][j] == 3:

        print("Ｓ", end="")

      elif maze[i][j] == 4:

        print("Ｇ", end="")

      elif maze[i][j] == 2:

        print("■", end="")

      elif maze[i][j] == 1:

        print("■", end="")

      elif maze[i][j] == 0:

        print("　", end="")

    print()

def grid():

  maze = []

  for i in range(HEIGHT):

    row = []

    for j in range(WIDTH):

      if i == 1 and j == 1:

        row.append(3)

      elif i == HEIGHT - 2 and j == WIDTH - 2:

        row.append(4)

      elif i == 0 or j == 0 or i == HEIGHT - 1 or j == WIDTH - 1:

        row.append(2)

      elif i % 2 == 0 and j % 2 == 0:

        row.append(1)

      else:

        row.append(0)

    maze.append(row)

  return maze

def create\_maze():

    maze = grid()

    for i in range(2, HEIGHT - 1, 2):

        for j in range(2, WIDTH - 1, 2):

            directions = []

            if i == 2:

                directions.append([i - 1, j])

            if maze[i+1][j] == 0:

                directions.append([i + 1, j])

            if maze[i][j-1] == 0:

                directions.append([i, j - 1])

            if maze[i][j+1] == 0:

                directions.append([i, j + 1])

            wall = random.choice(directions)

            maze[wall[0]][wall[1]] = 1

    return maze

HEIGHT = 11

WIDTH  = 11

display\_maze(create\_maze())

関数を1つずつ見ていきましょう。

**・display\_maze()**

　この関数は迷路を出力するプログラムです。maze[i]は各行、maze[i][j]は各マスを指定します。print(“Ｓ”, end=””)とすると、改行無しで文字列を出力することができます。また、print()は何も出力せずに改行をしてくれます。スタートマスは”Ｓ”、ゴールマスは”Ｇ”、壁は”■”、通り道は”　”（空白）で出力されます。

【プログラム例と解説】

|  |  |
| --- | --- |
| def display\_maze(maze):    for i in range(len(maze)):      for j in range(len(maze[i])):        if maze[i][j] == 3:          print("Ｓ", end="")        elif maze[i][j] == 4:          print("Ｇ", end="")        elif maze[i][j] == 2:          print("■", end="")        elif maze[i][j] == 1:          print("■", end="")        elif maze[i][j] == 0:          print("　", end="")      print() | display\_maze関数を宣言する  　iに0～len(maze)-1の整数を代入  　　jに0～len(maze[i])の整数を代入  　　　もしmaze[i][j]が3ならば  　　　　改行なしで”Ｓ”と出力  もしmaze[i][j]が4ならば  　　　　改行なしで"Ｇ"と出力  　　　もしmaze[i][j]が2ならば  改行なしで"■"と出力  　　もしmaze[i][j]が1ならば  改行なしで"■"と出力  　　　もしmaze[i][j]が0ならば  　　　改行なしで"　"と出力  改行する |

**・grid()**

　スタート・ゴールマス、壁の外壁、内側に1マスずつ設置をしてくれるプログラムです。外側はiが0・HEIGHT– 1、jが0・WIDTH-1のどれかの時に外壁を設置します。また、iが1かつjが1の時はスタートマス、iがHEIGHT-2かつjがWIDTH-2の時はゴールマスを設置します。内側の壁は、上記の場合を除いてiとjがどちらも2の倍数の時に設置をします。それ以外のマスは全て通り道となります。

　配列rowに行のデータを追加していき、それを二次元配列mazeに追加することによって、mazeを作成しています。

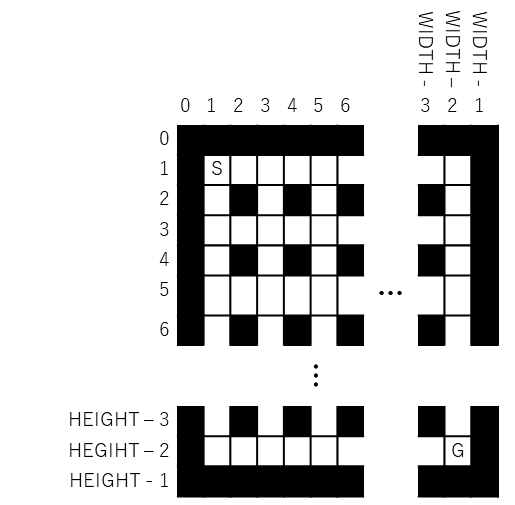


図12.14 迷路作成のイメージ図

【プログラム例と解説】

|  |  |
| --- | --- |
| def grid():    maze = []    for i in range(HEIGHT):      row = []      for j in range(WIDTH):        if i == 1 and j == 1:          row.append(3)        elif i == HEIGHT - 2 and j == WIDTH - 2:          row.append(4)        elif i == 0 or j == 0 or i == HEIGHT - 1 or j == WIDTH - 1:          row.append(2)        elif i % 2 == 0 and j % 2 == 0:          row.append(1)        else:          row.append(0)      maze.append(row)    return maze | grid関数の宣言  空の配列をmazeに代入  HEIHGT回処理を繰り返す  空の配列をrowに代入  WIDTH回処理を繰り返す  iが1、jが1ならば、  3をrowに追加  iがHEIGHT-2、jがWIDTH-2ならば、  4をrowに追加  iが0、jが0、iがHEIGHT-1、または、jがWIDTH-1であれば  2をrowに追加  iとjがどちらも偶数ならば、  1をrowに追加  上のどれにも当てはまらなければ、  0をrowに追加  mazeにrowを追加  mazeを返す |

　作成をしたら表示をしてみましょう。

HEIGHT = 11

WIDTH = 11

display\_maze(grid())

を全ての関数の下に記述することで、1マスずつ壁が配置されている迷路が出力されます。HEIGHTとWIDTHを別の奇数の値にすることで、他のサイズでも出力できるかを確かめましょう。

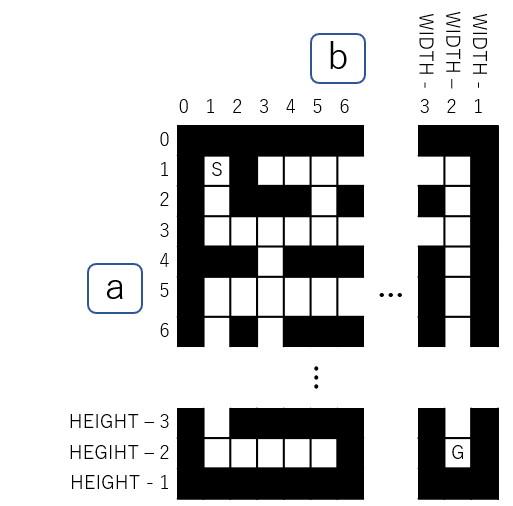


図12.15 aとbの位置関係

テーブル が含まれている画像

AI によって生成されたコンテンツは間違っている可能性があります。

図12.16 グリッドの出力

**・create\_maze()**

　create\_maze関数では、棒倒し法を用いて壁を作成し迷路を完成させます。directionsに候補となる箇所を代入して、そこからランダムに要素を選びます。[a,b]の形式で壁を配置する候補となる場所を選びます。aは縦の座標、bは横の座標を示します。

　迷路の特定のマスと、上下左右のマスの位置関係は次のようになっています。

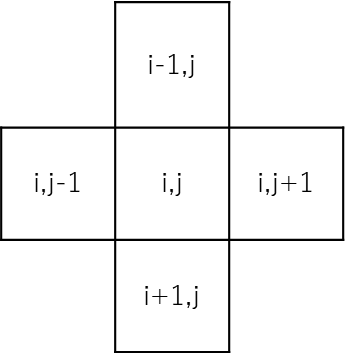


図12.17 マスの位置関係

【プログラム例と解説】

|  |  |
| --- | --- |
| import random  def create\_maze():      maze = grid()      for i in range(HEIGHT - 1):          for j in range(WIDTH - 1):              if i % 2 == 0 and j % 2 == 0 and i != 0 and j != 0:                directions = []                if i == 2:                    directions.append([i - 1, j])                if maze[i+1][j] == 0:                    directions.append([i + 1, j])                if maze[i][j-1] == 0:                    directions.append([i, j - 1])                if maze[i][j+1] == 0:                    directions.append([i, j + 1])                wall = random.choice(directions)                maze[wall[0]][wall[1]] = 1      return maze | randomモジュールのインポート  create\_maze関数の宣言  mazeにgrid関数を返す    HEIGHT-1回処理を繰り返す  WIDTH-1回処理を繰り返す  iとjが2の倍数かつ、0でなければ、    directionsに空の配列を代入  　　　　iが2なら  　　　　　directions配列に[i-1,j]を追加する  　　　　maze[i+1][j]が0なら  　　　　　directions配列に[i+1,j]を追加する  　　　　maze[i][j-1]が0なら  　　　　　directions配列に[i, j-1]を追加する  　　　　maze[i][j+1]が0なら  　　　　　directions配列に[i, j+1]を追加する  　　　　directionsからランダムに選びwallに代入  　　　　maze[wall[0]][wall[1]]に1を代入  　mazeを返す |

　これで、迷路作成プログラムが完成しました。

HEIGHT = 11

WIDTH = 11

display\_maze(create\_maze())

を全ての関数の下に記述することで、プログラムが実行されるかを確かめてみましょう。

* 1. 迷路の解法プログラム

　迷路の解法プログラムを作成しましょう。迷路の解法プログラムでは、grid()・create\_maze()・display\_maze()に加え、search()・solve\_maze()関数を用いてプログラムを作成します。display\_maze関数は少し改良を加えます。

* search () … 迷路を探索するプログラム
* solve\_maze() … 迷路を解くプログラム

【プログラム例】

import random

def display\_maze(maze, path):

  for i in range(len(maze)):

    for j in range(len(maze[i])):

      if [i, j] in path:

        print("＃", end="")

      elif maze[i][j] == 3:

        print("Ｓ", end="")

      elif maze[i][j] == 4:

        print("Ｇ", end="")

      elif maze[i][j] == 2:

        print("■", end="")

      elif maze[i][j] == 1:

        print("■", end="")

      elif maze[i][j] == 0:

        print("　", end="")

    print()

def grid():

  ...

def create\_maze():

  ...

def search(maze, maze\_search, pos):

    d = [[0, 1], [1, 0], [0, -1], [-1, 0]]

    for i in d:

        next\_pos = [pos[0] + i[0], pos[1] + i[1]]

        maze\_condition = maze[next\_pos[0]][next\_pos[1]]

        if maze\_condition == 4:

            maze\_search[next\_pos[0]][next\_pos[1]] = 1

            return [1, []]

        if maze\_condition == 0 and maze\_search[next\_pos[0]][next\_pos[1]] == 0:

            maze\_search[next\_pos[0]][next\_pos[1]] = 1

            result = search(maze, maze\_search, [next\_pos[0], next\_pos[1]])

            if result[0] == 1:

                path = result[1]

                path.insert(0, next\_pos)

                return [1, path]

    return [0, []]

def solve\_maze(maze):

    maze\_search = []

    for i in range(HEIGHT):

      row = []

      for j in range(WIDTH):

        row.append(0)

      maze\_search.append(row)

    result = search(maze, maze\_search, [1, 1])

    if result[0] == 0:

        return "cannot\_goal"

    return result[1]

HEIGHT = 11

WIDTH = 11

maze = create\_maze()

path = solve\_maze(maze)

display\_maze(maze, solve\_maze)

**・display\_maze()**

　引数pathを追加して、4,5行目を追加します。pathは正解の経路を示す引数で、[[1.1],[1,2],[1,3],[2,3],…]のように座標を記録しています。経路は”＃”で示します。

【プログラム例】

def display\_maze(maze, path):

  for i in range(len(maze)):

    for j in range(len(maze[i])):

      if [i, j] in path:

        print("＃", end="")

      elif maze[i][j] == 3:

        print("Ｓ", end="")

      elif maze[i][j] == 4:

        print("Ｇ", end="")

      elif maze[i][j] == 2:

        print("■", end="")

      elif maze[i][j] == 1:

        print("■", end="")

      elif maze[i][j] == 0:

        print("　", end="")

    print()

**・solve\_maze()**

　solve\_maze関数は、迷路を探索してどこにゴールがあるのかを調べるプログラムです。このプログラムの概要は以下の通りです。

**引数**

* maze … 迷路状況
* maze\_search … 迷路の探索を行った場所
* pos … 迷路で探索している場所

maze\_searchは下の図のように、探索をすでにしている箇所を1、まだ探索を行っていない箇所を0として記録する引数です。

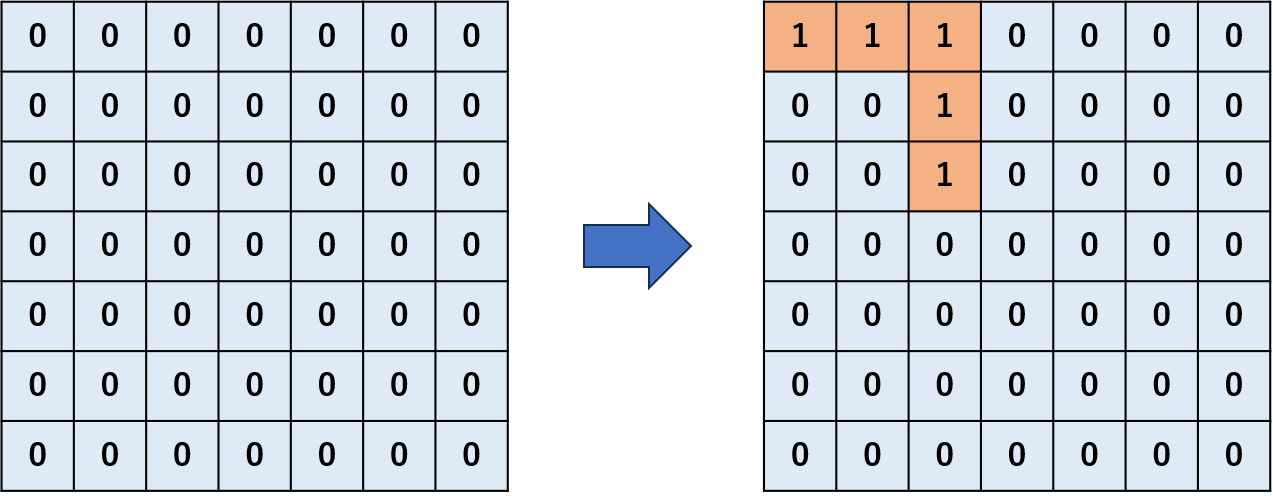


図12.18 maze\_search変数のイメージ図

**プログラム**

① 移動できる先を表す変数dの宣言（2行目）

1つ目の要素の[0,1]は1つ右のマス、[1,0]は下のマス、[0,-1]は左のマス、[-1,0]は上のマスを表します。

② 移動先を変数next\_pos、移動先の状況をmaze\_conditionに代入（3～5行目）

　3行目では、移動できる先を順にiに代入します。このプログラムにより、現在のマスの上下左右のマスを、右・下・左・上の順で確認します。引数posには現在のマスが代入されています。4行目では、移動先のマスの座標を変数next\_posに代入します。maze\_conditionには移動先が、壁・通り道・スタート・ゴールのどれのマスなのかが代入されます。

③ 移動先がゴールだった場合（6～8行目）

　maze\_conditionが4の場合、移動先のマスがゴールだということになります。7行目で移動先のmaze\_searchを1にすることによって、探索をしたことを記録します。そして、戻り値を指定します。1と空の配列[]が返されます。1はゴールしたことを示します。空の配列はどのように移動したのかを示す経路を格納します。

④ 移動先が通り道かつ未探索の場合（9～12行目）

　まず9行目で、移動先のmaze\_searchを1にすることによって探索済みであることを記録します。そして、result変数にsearch関数の戻り値を代入します。search変数は、maze・maze\_searchと、posには移動先の座標を指定します。result変数には、2つの情報が返されます。result[0]には、ゴールができたかが代入されます。0の時はゴールできない（行き止まりになる）、1の時はゴールできます。result[1]はどのように移動したのかを示す経路が代入されます。

⑤ 探索をした結果ゴールだった場合（13～16行目）

　result[0]が1の時、ゴールをしています。result[1]は経路を表し、それを変数pathに代入します。そして、移動先の座標を追加して、戻り値を返します。戻り値には、ゴールできたことを示す1と経路を示すpathを指定します。

【プログラム例】

def search(maze, maze\_search, pos):

    d = [[0, 1], [1, 0], [0, -1], [-1, 0]]

    for i in d:

        next\_pos = [pos[0] + i[0], pos[1] + i[1]]

        maze\_condition = maze[next\_pos[0]][next\_pos[1]]

        if maze\_condition == 4:

            maze\_search[next\_pos[0]][next\_pos[1]] = 1

            return [1, []]

        if maze\_condition == 0 and maze\_search[next\_pos[0]][next\_pos[1]] == 0:

            maze\_search[next\_pos[0]][next\_pos[1]] = 1

            result = search(maze, maze\_search, [next\_pos[0], next\_pos[1]])

            if result[0] == 1:

                path = result[1]

                path.append(next\_pos)

                return [1, path]

    return [0, []]

**・solve\_maze()**

**引数**

・maze … 迷路の状況

**プログラム**

① 迷路探索済みかを示すmaze\_search変数の作成（2～7行目）

　rowに行のデータを作成し、それをmaze\_searchに追加することによって作成しています。

② 結果を変数resultに代入し、戻り値にする（9,10行目）

　search関数を呼び出します。迷路の状況を示すmaze、探索済みかを記録するmaze\_searchを指定します。また、スタートタイルの位置は[1,1]なので引数posにはそれを指定します。

　result[1]に経路が代入されるので、それを返します。

プログラムを実際に動かしてみましょう。全ての関数の下に以下のプログラムを記述して、動作を確認してみましょう。上手く動作したら、HEIGHT・WIDTHの値を変更して迷路のサイズを変えてみましょう。

【プログラム例】

def solve\_maze(maze):

    maze\_search = []

    for i in range(HEIGHT):

      row = []

      for j in range(WIDTH):

        row.append(0)

      maze\_search.append(row)

    result = search(maze, maze\_search, [1, 1])

    return result[1]