アプリ
プログラミングシート

ソーメンタイマー
(Python版)

学習目標

観点	学習目標
知識・技能	• 指定時間ごとに処理のかたまりを実行する仕組みを理解する。 • 「ミリ」などの接頭語を理解する。 • 値を格納し、更新し、参照して、変数の利用方法を理解する。 • 条件（残り時間の値）によって、異なった処理を実行する条件分岐制御構造を理解する。 • HTMLを用いて、画面の表示内容を指定できることを理解する。 • CSSを用いて、画面の表示方法を指定できることを理解する。 （制作） • 一定時間経過ごとに処理を呼び出し、残り時間に応じて実行する処理を分岐させる構造を利用して、独自のタイマーアプリケーションを作成することができる。 • プログラムの中で使用する音声ファイルを作成・編集することができる。
思考力・判断力・表現力	（制作） • サンプルのアプリケーションを参考にしながら、カウントダウンするタイマーの題材を考え、題材に適合する画面表示・文字表示や、音声データを作成して、表現している。
態度	（制作） • HTML上の文字列、条件分岐するプログラム、音声ファイルなど、様々な素材を適切な組み合わせを粘り強く考えている。 • 作成したアプリケーションが適切に動作することを、丁寧にテストして、確かめている。

単元の流れ

コマ	内容	狙い
1	タイマーアプリケーションの動作を確認し、カスタマイズ方法を学ぶ	• アプリケーションが、HTML、プログラム、音声ファイルなど様々な要素の組み合わせでできていることを学ぶ。 • 指定した時間ごとに処理のかたまりを実行する仕組みを学ぶ。「ミリ秒」の指定を通じて、接頭語について知る。 • 変数countの値を使って、タイマーの初期の残り時間を決めていること、これを一定時間経過ごとに減らすことでタイマーを実現する仕組みを理解する。 • 残り時間ごとの処理を変更することを通じて、条件分岐制御構造を学ぶ。
2～	（オプション） オリジナルのタイマーアプリケーションを制作する	• アプリケーションのタイトルや、スタートボタンの文字表示を変更する手順を通じて、HTMLの役割を学ぶ。 • 表示方法を変更する手順を通じて、CSSの役割を学ぶ。 • タイマーアプリケーションを企画・設計する。何のための時間を測るか、経過時間ごとにどのような音声を流すかなどを考える。 • アプリケーションを制作する。音声ファイル、HTMLファイル、CSSファイル、プログラムなどを作成・編集する。 • 意図通りに制作できたかどうかテストし、必要なら修正を加える。

（留意事項）

1コマ目の指導

過程	内容
導入	「1秒経過するごとに、指定した関数（処理のかたまり）を実行する」という仕組みを利用した、カウントダウンするタイマーのアプリケーションを学ぶことを伝える。 Monaca Educationにログインし、APSソーメンタイマー（Python版）をインポートする。
展開1	アプリケーションを動かし、動作を確認する。 タイマーは、1分20秒（80秒）を測る。 スタートボタン押下直後、残り60秒、残り30秒、残り10秒、残り0秒（80秒経過）時に、それぞれ音声を再生する。
展開2	フローチャートを簡単に確認した後、カスタマイズを行なって、仕組みを学ぶ。 カスタマイズ①：カウント（変数count）を表示する。1秒経過するごとに表示されることを確認する。 カスタマイズ②：タイマーの全体の時間（count）を決める。デフォルトは80だが、これを120に変えてみる。 変更した時間（count）は、次のカスタマイズの前に戻す。 カスタマイズ③：処理を繰り返す間隔を、5秒（5000ミリ秒）に変更する。デフォルトは1秒（1000ミリ秒）。参考資料として接頭語の一覧（※各社の教科書の資料ページにも掲載されている）を確認させる。 変更した間隔は、次のカスタマイズの前に戻す。 カスタマイズ④：条件分岐を追加する。カウントが残り半分になったときに、画面に「残り半分！」と表示する。
（オプション） 展開3	以下のカスタマイズでは、HTMLやCSSの機能を学ぶ。 作品制作の準備として、２コマ目以降に行わせることができる。 カスタマイズ⑤：HTML（index.html）を編集して、アプリケーションのタイトルを変える。 カスタマイズ⑥：CSS（style.css）を編集して、アプリケーションの色使いを変える。
まとめ	タイマーアプリケーションにカスタマイズを加えることを通じて、仕組みを理解したこと、プログラミングによって、下記のようなことができることを学んだことを確認する。 • 一定時間ごとに決まった処理のかたまりを実行する • 条件によって処理を切り替える（条件分岐） その他、接頭語を学んだ。 • 時間の単位に、接頭語をつけることで、大きさを指定できる

2コマ目以降の指導

過程	内容
導入	「1秒経過するごとに関数（処理のかたまり）が実行される」という仕組みを利用して、カウントダウンするタイマーのアプリケーションを作ることを説明する。 1コマ目で学習した、アプリケーションの仕組みと、基本的なカスタマイズ法を振り返る。 • 一定時間経過ごとに処理のかたまりを繰り返し実行する仕組みについて、処理を実行するまでの時間を変更する • タイマーの全体の時間を指定する • 処理を繰り返す時間を指定する • 条件分岐を追加する
展開1	以下のカスタマイズでは、HTMLやCSSの機能を学ぶ。 作品制作の準備として、２コマ目以降に行わせることができる。 カスタマイズ⑤：HTML（index.html）を編集して、アプリケーションのタイトルを変える。 カスタマイズ⑥：CSS（style.css）を編集して、アプリケーションの色使いを変える。
展開2	タイマーアプリケーションの企画・設計を行う。 • 何の時間を測るタイマーにするか • どんな音声メッセージを流すか（どんな音声ファイルが必要か） • アプリケーションの外観（アプリケーションのタイトルや、色使い）をどうするか など
展開3	企画・設計したアプリケーションを制作する。 音声を録音・編集したり、HTML・CSSを編集したり、プログラムを作成したりする。 制作したアプリケーションをテストし、意図した通りに動作するか検証する。 意図通りでない場合、アプリケーションを修正する。
まとめ	学習した事柄を確認する。 （オプション） • 制作したアプリケーションに盛り込んだ機能や工夫をワークシートなどに記録する。 • 作成したアプリケーションをMonaca EducationのWeb共有機能などを用いて、クラスや学年で共有し、相互評価・コメントを行う。

APSソーメンタイマー（Python版）

アプリの概要

スタートボタンを押すと、１秒ずつ残り時間が減っていく。

スタート直後と、区切りのタイミング（残り1分、30秒、10秒、ゼロ秒）で音声による指示が流れる。

学習内容

要素技術	内容
定期実行	• 「指定された時間が経過したら、指定した処理（関数）を実行」という仕組みを使う • 「スタートボタンが押されたら、関数clickStart()を実行」と設定する。関数clickStart()の中から、関数main()を呼び出す • 関数main()の最後には、「1秒後に関数main()を実行」というプログラムを書いておく • 上記の設定とプログラムの結果、「スタートボタンが押されたら、以後、1秒経過するごとに関数main()を実行する」という定期実行の仕組みになる
タイマー動作	• 最初に、変数countに値80を代入しておく • 関数main()が呼び出されるたびに、変数countから1引いて、その値を変数countに代入するようにする（例：初めて関数main()呼び出されたら、80から1引く。その結果の79を、変数countに代入する） • 関数main()が呼び出されるたびに、変数countの値を調べ、60なら「残り時間が60秒」と判定し、音声を流すなどの処理を行う（変数countの値が30、10、0の場合も同様。条件によって処理を変える仕組み（制御構造）を条件分岐構造と呼ぶ） • 関数main()が80回呼び出されたら、変数countはゼロになる。タイマーとして動作を終了する
条件分岐	• 変数countの値が60と等しい場合、残り60秒の場合の音声を流す • 変数countの値が30と等しい場合、残り30秒の場合の音声を流す • 変数countの値が10と等しい場合、残り10秒の場合の音声を流す • 変数countの値が0と等しい場合、終了の場合の音声を流す。その他、終了するための処理を実行する
画面表示（HTML/CSS）	• （HTML）画面にアプリケーションのタイトル（ソーメンタイマー）や、残り時間表示を行っている • （CSS）画面の要素の表示方法を決めている（画面の背景の色、文字の色、要素の左右中央配置など）

ソーメンタイマー（Python版）を動かしてみよう(1)

サポートページから
ソーメンタイマー（Python版）
をクリック

Monacaプロジェクトとして
インポートして、IDEで開く

残り時間「01:20」と
スタートボタンが表示される

プログラムを読んでみよう
（※index.html）

```
<!-- タイトル -->
<h1 id='app-title'>ソーメンタイマー</h1>                    アプリのタイトル

<!-- 残り時間 -->
<div id='timer-area'>
    <span id='timer'>00:00</span>                          残り時間
</div>

<!-- ボタン -->
<div id='control-area'>
    <input id="startButton" type="button" value="スタート" />     ボタン
</div>

<!-- print()で文字を表示する領域 -->
<div id='contents'></div>
```

（参考）index.html ボディ（<body>）全体

```
<body onload='brython();'>
  <audio id="start-mp3" preload>
    <source src="audio/start.mp3" type="audio/mp3">
  </audio>
  <audio id="s60" preload>
    <source src="audio/s60.mp3" type="audio/mp3">
  </audio>
  <audio id="s30" preload>
    <source src="audio/s30.mp3" type="audio/mp3">
  </audio>
  <audio id="s10" preload>
    <source src="audio/s10.mp3" type="audio/mp3">
  </audio>
  <audio id="end-mp3" preload>
    <source src="audio/end.mp3" type="audio/mp3">
  </audio>

  <!-- タイトル -->
  <h1 id='app-title'>ソーメンタイマー</h1>

  <!-- 残り時間 -->
  <div id='timer-area'>
      <span id='timer'>00:00</span>
  </div>

  <!-- ボタン -->
  <div id='control-area'>
      <input id="startButton" type="button" value="スタート" />
  </div>

  <!-- print()で文字を表示する領域 -->
  <div id='contents'></div>
</body>
```

音声ファイルにプログラムから呼び出すための名前（※id=""の部分）を付けている

ID"end-mp3"は、フォルダaudioにあるファイルend.mp3を指す

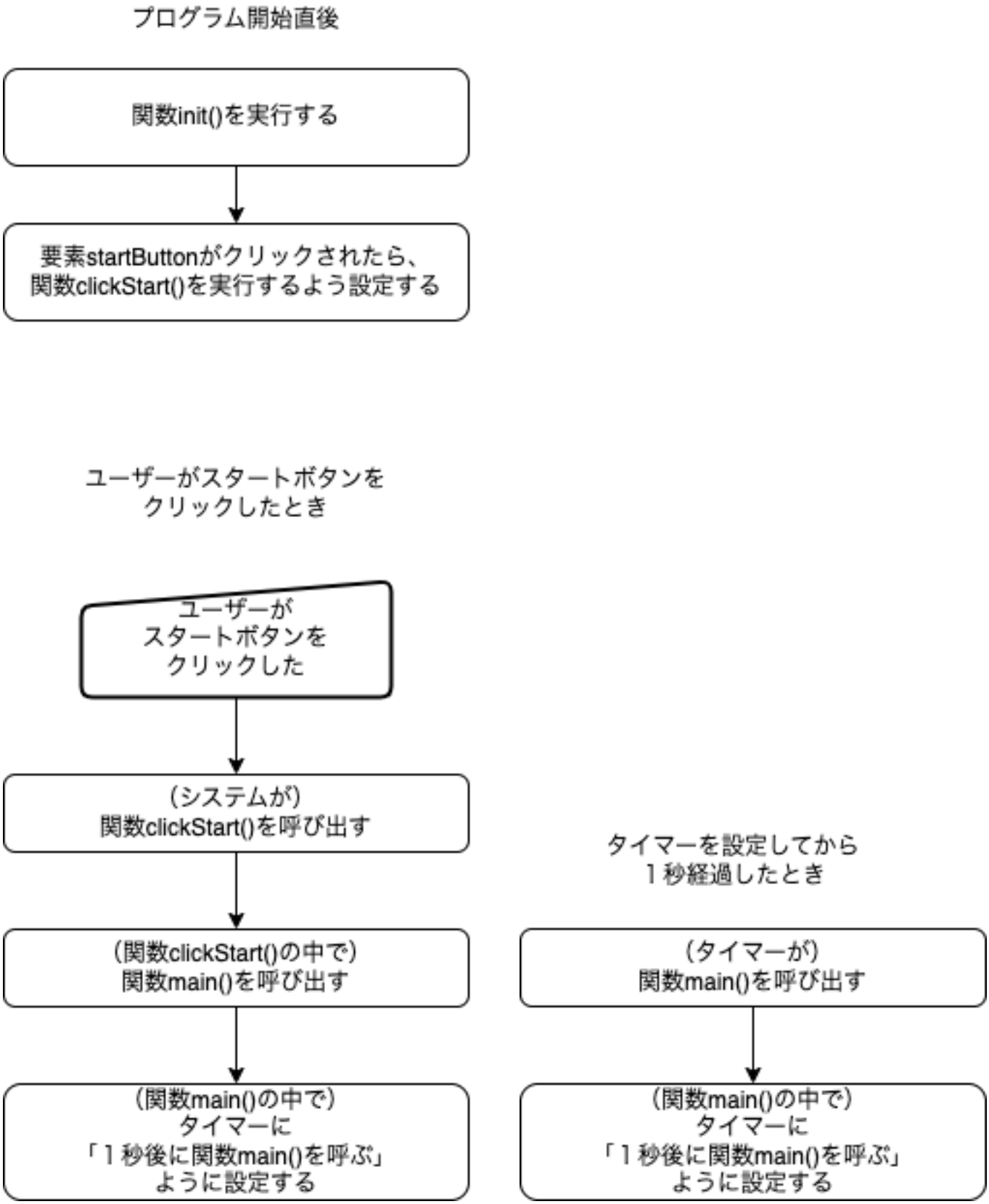
アプリのタイトル

残り時間表示。プログラムで1秒おきに書き換えている

プログラムの中では"startButton"。画面には「スタート」ボタンと表示される

print()で値を出力する箇所

フローチャート：１秒ごとに同じ処理を繰り返す仕組み



ソーメンタイマー（Python版）を動かしてみよう(2)

スタートボタンを
クリックすると、
音声が流れる

1秒ごとに残り時間が
減っていく。
区切りで音声が流れる

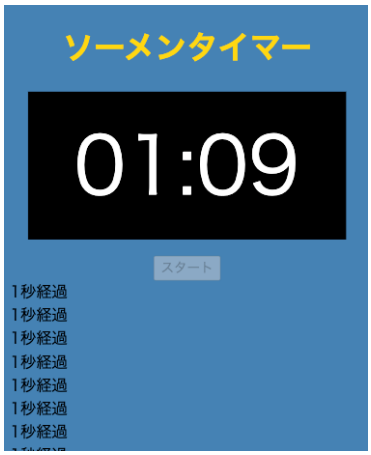
残り時間が0になると
音声が流れ、最初の状態に戻る

プログラムを読んでみよう
（※python.py）

python.pyの54行目から下

```
# スタートしたときに呼び出される処理。その後、1秒おきに実行される
def main():                                                    関数（処理のまとまり）
  global first
  global count
  if first == True: # 開始時点

  …                                          （中略）

  if count >= 0:                    もし、変数countが0以上なら
    timer.set_timeout(main, 1000) # 1秒後（1000ミリ秒後）にmainを実行する
  else:
    del document["startButton"].attrs["disabled"] # スタートボタンを押せるようにする
    init()


def clickStart(ev):                                  関数clickStart()が呼び出されたら、
  main()                                                       関数main()を呼び出す

##############################################
# ！この下の行は消さないで！
##############################################
init() # 初期化処理を実行する                                  初期化処理の関数init()を呼び出す
# HTMLに書いた"startButton"がクリックされたら、関数mainを呼び出すように設定する
# （※ボタンが押されたときに実行するよう仕掛けをするだけで、実際にmainは実行はしない）。
document["startButton"].bind("click", clickStart)              HTMLの要素startButtonが
                                  クリックされたら、関数clickStart()を呼び出すように設定する
```

カスタマイズ①　1秒ごとにメッセージを追加表示する



スタートボタンを押したら、
1秒おきに
画面下にメッセージを表示する

プログラムを変更

```
time = count2time(count) # 残り秒数を「00:00」形式にする
timer_field.innerHTML = time # 残り秒数の表示を変える
count = count - 1

print("1秒経過")                    関数main()の終盤に、print()を追加する

if count >= 0:
   timer.set_timeout(main, 1000) # 1秒後（1000ミリ秒後）にmainを実行する
…
```

関数main()の中に、命令print()を、1回書くだけ
→それにも関わらず、1秒ごとに画面に文字が追加されていく

なぜそんな結果になるか？
→関数main()が、1秒おきに呼び出されているから

※関数print()は、HTMLのcontentsという要素の中に文字列を追加している

カスタマイズ② タイマーの時間を変える

スタート時点での80秒（01:20）を、

120秒（02:00）にする

プログラムを変更

```
# 初期化処理を定義する
def init():
  global timer_field
  global sounds # 音声データ
  sounds["startSound"] = document["start-mp3"]
  sounds["s60Sound"] = document["s60"]
  sounds["s30Sound"] = document["s30"]
  sounds["s10Sound"] = document["s10"]
  sounds["endSound"] = document["end-mp3"]
  global first # 初回実行かどうかを管理するフラグ
  first = True # 初期化する
  global count # 残り時間数                          count = 80を
  count = 120 # 残り秒数を80に設定している          count = 120にする
  time = count2time(count) # 残り秒数を"00:00"形式に変換する
  timer_field.innerHTML = time # 残り秒数の表示を変える
```

変数count：残り秒数を格納しておく入れ物

関数init()は、プログラムが始まって、最初に実行される処理のかたまり。
この関数の中の、count=80の行を、count=120にする。

カスタマイズ③ 繰り返す間隔を変える

5秒経過すると、

タイマーの表示が1秒ずつ減っていく

プログラムを変更

```
if count >= 0:
    timer.set_timeout(main, 5000) # 1秒後（1000ミリ秒後）にmainを実行する
```

1000 → 5000

※Brython環境でのタイマー設定の方法

timer.set_timeout(<呼び出す関数名>,　何ミリ秒後)

ミリ秒（ms）とは

1秒の千分の1＝1ミリ秒。

「ミリ」は接頭辞で、「千分の1」（10^{-3}）を意味する。
他の例：

- mm（ミリメートル）は、1mの千分の1
- mL（ミリリットル）は、1Lの千分の1

カスタマイズ④ 残り半分になったら メッセージを表示する

残り半分（40秒）になったら、
「あと半分！」
というメッセージを画面に表示する

プログラムを変更

```
elif count == 60: # 60秒前
    sounds["s60Sound"].play()
elif count == 40: # 40秒前
    print("あと半分！")
elif count == 30: # 30秒前
    sounds["s30Sound"].play()
```

elif 条件式:で改行して、
（インデントの後）print()

elif count == 60: の条件分岐の後、
elif count == 30: の条件分岐の前に、elif count == 40: の分岐を追加する

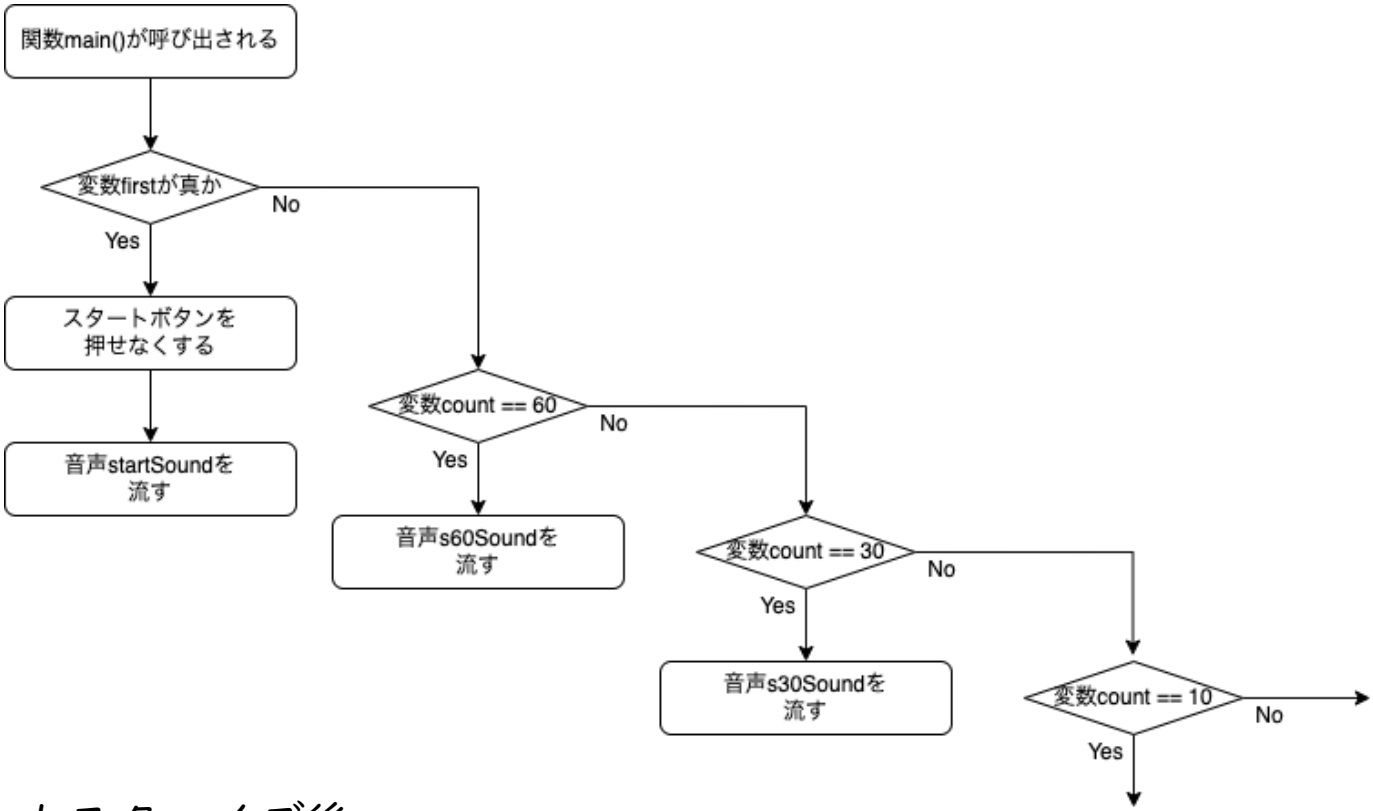
print(”あと半分!”)の行は、タブキーを押して空白を入れる。
これを字下げ（インデント）と呼び、指定の条件に合っている時、どの行を実行するか範囲を決めている。

※プログラムの実行中に、count == 30とcount == 40が同時に起きることは無いため、残り時間の大きい方から並べる必要はない。順番を逆にしても、プログラムは問題なく動作する（40秒前には40秒前の処理が、30秒前には30秒前の処理が実行される）。

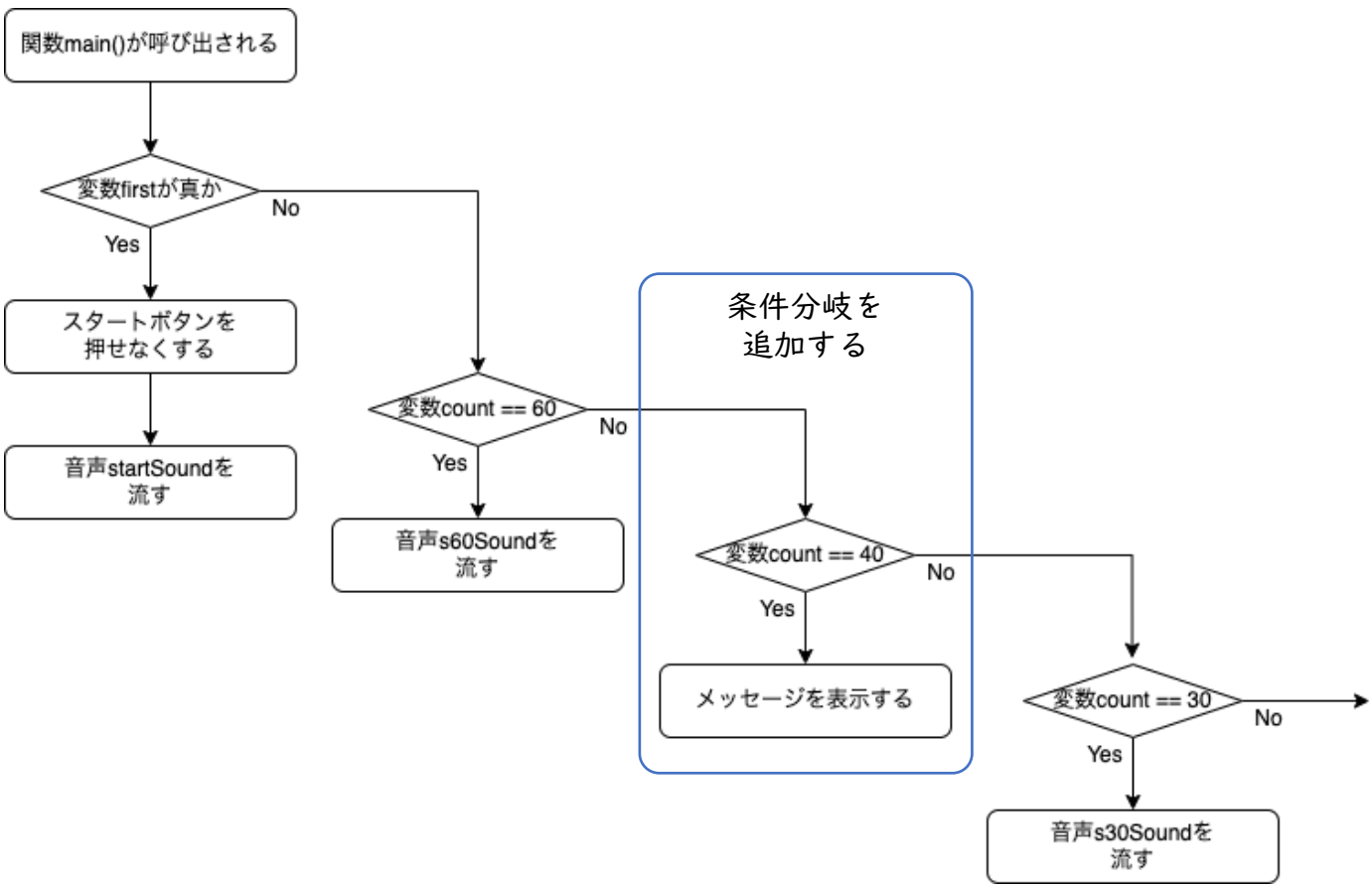
後からプログラムを見る人間のために、分かりやすい並び順にしている。

フローチャート：半分経過した時の処理を追加する

カスタマイズ前



カスタマイズ後



カスタマイズ⑤ アプリケーションのタイトルを変える

アプリケーションのタイトル部分を
「カップラーメンタイマー」に
変更した。

※オプション
タイマーの残り時間を3分にした。

プログラムを変更
(※index.html)

```
<!-- タイトル -->
<h1 id='app-title'>カップラーメンタイマー</h1>
```

index.htmlの
<h1>タグの中身を書き換える

カスタマイズ⑥ 画面の要素の色を変更する

画面の背景の色と、
アプリのタイトルの文字の色を変更

プログラムを変更
(※style.css)

```
body {
  background-color: orange;
}
```

要素bodyの中の
background-color: 色名 で、
背景の色を指定する

```
#app-title {
  text-align: center;
  color: saddlebrown;
}
```

要素"#app-title"の中の
color: 色名 で、
文字の色を指定する

CSSファイルの中のbodyは、index.htmlのタグ<body>を指している。
上の指定は、
「タグ<body>を画面に表示するとき、背景色（background-color）は、orangeにしなさい」
という意味である。

CSSファイルの中の#app-titleは、index.htmlの次の要素を指している。

```
<!-- タイトル -->
<h1 id='app-title'>カップラーメンタイマー</h1>
```

上の指定は、
「id="app-title"の要素を画面に表示するとき、文字色（color）は、saddlebrownにしなさい」
という意味である。

接頭辞

小さな量の接頭辞

記号	接頭辞	読み方	大きさ
m	milli	ミリ	10^{-3}。1000分の1
μ	micro	マイクロ	10^{-6}。100万分の1
n	nano	ナノ	10^{-9}。10億分の1
p	pico	ピコ	10^{-12}。1兆分の1

大きな量の接頭辞

記号	接頭辞	読み方	大きさ
k	kilo	キロ	10^{3}。1000倍
M	mega	メガ	10^{6}。100万倍
G	giga	ギガ	10^{9}。10億倍
T	tera	テラ	10^{12}。1兆倍

（参考） 音声を変えるには

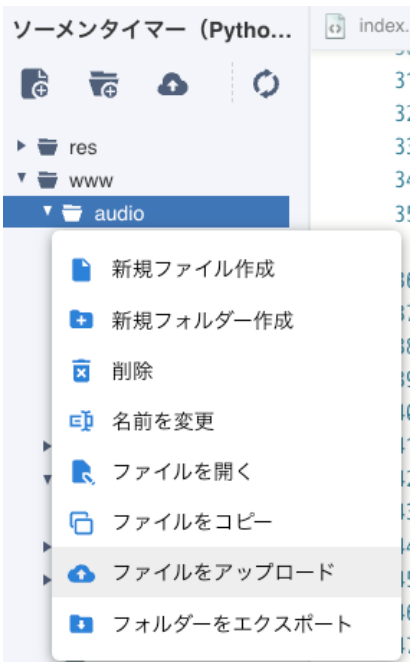
index.html	python.py
タグ<audio> ファイルに IDを割り当てる	関数init() IDを指定して取り出す 関数main() 条件分岐の中で使う

音声ファイルを作成する

PCやタブレット、スマートフォン等に付属する録音機能・アプリを用いて、音声を録音する。Monaca Educationで開発を行う機器（PCやタブレット）と別の機器で録音した場合、ファイルをMonaca Educationで開発を行う機器に移す

ファイルをMonaca Educationのプロジェクトにアップロードする

以下の例では、ファイル名をmy.mp3としている



index.htmlでタグ<audio>を使い、ファイルにIDを割り当てる

```
<audio id="mySound" preload>
    <source src="audio/my.mp3" type="audio/mp3">
</audio>
```

プログラムでは割り当てたIDを使って音声ファイルにアクセスするようにする

関数init()

```
sounds["mySound"] = docment["mySound"]
```

関数main()の中の音声を流す箇所

```
sounds["mySound"].play()
```

確認テスト

問題	回答
1秒の1000分の1（10^{-3}）を表すために使われる接頭語（単位の名前の前に付ける語）を書いてください。 また、読み仮名を（）で囲んで書いてください。	m（ミリ）
プログラム言語Pythonの中で、「真である」ことを表すための値は何か。書きなさい。 なお、「偽である」ことを表すための値はFalseである。	True

「変数countの値が10と等しければ『A』を実行する、そうでなく変数countの値が20と等しければ『B』を実行する」という条件分岐を、次のように書くとします。
`if count == 10:`
`  A`
`elif count == 20:`
`  B`
 	
このとき、次のような条件分岐をするプログラムを書いてください。
 	
「変数aの値が2と等しければ『C』を実行する。そうでなく変数aの値が1と等しければ『D』を実行する」
	`if a == 2:`
	`  C`
	`elif a == 1:`
	`  D`

「変数countの値が10と等しければ『A』を実行する、そうでなく変数countの値が20と等しければ『B』を実行する」という条件分岐を、次のように書くとします。
`if count == 10:`
`  A`
`elif count == 20:`
`  B`
 	
ここで、変数countの値が30のとき、どのような振る舞いになるでしょうか。正しいものを選んでください。
	• 処理Aが実行される
	• 処理Bが実行される
	• 何も実行されない

記述問題は実際のプログラムで回答しても構いません。
また、プログラム言語の文法に厳密に従っていなくても、考え方が正しければ正解とします。