Pythonの勉強

環境設定、式、命名規則

学習目標

- Python環境設定 (Python Setup)
- 変数 (Variables)
- 命名規則 (Naming Conventions)
- 組み込み関数 (Built-in functions)
- 式 (Expressions)
- エラーの種類 (Types of Errors)

環境設定

Python 3（最新版）
ダウンロード: https://www.python.org/downloads/

Pycharm IDE
無料コミュニティーバージョン: https://www.jetbrains.com/

Python Online Complier

オンラインコンパイラ： https://www.programiz.com/python-programming/online-compiler/

コメント

コード内のコメントは、プログラムを読む人向けの説明書きです。

Pythonのインタープリタには無視され、実行されることはありません。

コメントは、コードの説明やメモとして役立ちます。

コメントは # で始まり、その行の最後まで続きます。

```
# this is a comment
print("hello world")  # this is an inline comment
```

コードスタイル

Pythonのコードは、他の人が読んでもわかりやすいように書くことを心がけ、一貫性を持たせて書きましょう。

https://www.python.org/dev/peps/pep-0008/

PEP：Pythonコミュニティによって作成されたスタイルガイド

https://google.github.io/styleguide/pyguide.html

Google Python Style Guide: Googleが内部で使用するために作成したスタイルガイド

変数とデータタイプ

変数は、コンピュータのメモリに保存された値（データ型を持つ）を表す名前です。
値には以下のような異なる型があります：

- 文字列（String）　“引用符”または‘引用符’で囲まれた任意のもの、例えば「Hello World!」や「this」
- 整数（Integer）　10、5、101、-66、0などの整数
- 浮動小数点数（Floating point）　11.65、-0.001、1200.0などの小数

- 代入文（assignment statement）を使用して新しい変数を作成できます。
x = 5;
代入文は、イコール記号（=）を使用して変数に値を与えます。
nickname = “RIN”

変数の例

変数の型は、変数に格納されている値に基づきます。

```
x = 5                                   # integer
x = -10                                 # integer
first_name = "susan"                    # string
last_name = 'smith'                     # string
biweekly_salary_usd = 1200.05           # floating point
```

変数のタイプ

値の型が変わると、変数の型も変わります。

```
x = 10                  # integer
x = 10.156              # floating point
x = "Hello World"       # string
```

変数の型は、組み込み関数を使用して変更できます。

```
x = 4               # integer
y = float(x)        # Converts 4 to 4.0
a = 5.6             # float
b = int(a)          # Converts 5.6 yo 5
z = 7.8             # float
z_str = str(z)      # Converts 7.8 to "7.8"
```

命名規則

意味のある名前を選んで、変数の用途を文書化してください。
文字、数字、アンダースコア文字を含むことができますが、数字では始められません。
Pythonのキーワードを使用することはできません。

```
and       del       from      None      True
as        elif      global    nonlocal  try
assert    else      if        not       while
break     except    import    or        with
class     False     in        pass      yield
continue  finally   is        raise     async
def       for       lambda    return    await
```

命名規則

変数、モジュール、関数名には lower_snake_case を使用する:

my_input, first_name, last_name, print_result

定数（値が変わらないもの）には UPPER_CASE を使用する:

PI = 3.14, GRAVITY_M_SEC_SQ = 9.8

クラスには UpperCamelCase を使用する（レッスン11で扱う）:

BankAccount, CoinFlip, Animal

関数

組み込み関数 input() を使用したコンソール（キーボード）からの入力

コンソールからの入力をユーザーに促します

```
print('Enter something')  # Print a prompt
my_input = input()  # User input is stored as a string in variable my_input
print(my_input)  # Output the user input back to the console
print("You entered:", my_input)  # Output the user input with some leading text
```

コンソールはこのように表示されます

```
Enter something
This is a test
This is a test
You entered: This is a test
```

関数

input()は、コードを短縮するためのプロンプトも受け取ることができます。

```
# User input is stored as a string in variable my_input
my_input = input("Enter something \n")
print(my_input)  # Output the user input back to the console
print("You entered:", my_input)  # Output the user input with some leading text
```

コンソール

```
Enter something
This is another test
This is another test
You entered: This is another test
```

関数

print()を使用してコンソール（画面）に出力する

```
# Each print statement ends with a new line
print("hello there,")
print("General")
print("Kenobi")
```

```
hello there,
General
Kenobi
```

```
#  Each print statement keeps output on the same line separated by a space
print("hello there,", end=' ')
print("General", end=' ')
print("Kenobi", end=' ')
```

```
 hello there, General Kenobi
```

関数

すべてのものはPythonにおいてオブジェクトであり、整数や文字列のような変数も含まれます。オブジェクトについては後ほど詳しく説明します。
type()は、オブジェクトのデータ型を返します。

```
x = 1
print(type(x))
```

```
<class 'int'>
```

id()は、オブジェクトのIDを返します。このIDはプログラム内でオブジェクトを一意に識別する整数です。

```
x = 1
print(id(x))
```

```
140727023638176
```

算術演算子

式は新しい値に評価される項の組み合わせです。
変数 – 保存された値で、時間の異なる点で異なる値を持つことができます。
リテラル – 2などの特定の値。
演算子 – 組み込み計算を実行する記号。

Pythonの算術演算子:

- 加算 (x + y)
- 減算 (x - y)
- 乗算 (x * y)
- / 除算 (x / y)
- // 整数除算 (結果は整数になります; 小数部分は切り捨てられ、四捨五入されません)
- ** 指数 (xのy乗)
- % 割り算の余り

算術演算子

例

```
x = 5
y = 6 + x          # y is now 11
z = y - 1          # z is now 10
a = y * z          # a is now 110
b = 100
c = b / 5          # c is now 20
base = 2
exp = 3
value = base ** exp  # value is now 8
```

算術演算子

例

```
x = 5.5
y = 6.1 + x      # y is now 11.6
z = y - 1.1      # z is now 10.5



a = 5.6
b = 11
c = int(a) + b  # c is evaluated as 5 + 11 = 16
```

エラーの種類

構文 (Syntax) – プログラミング言語のルール

構文エラー (Syntax Error) – プログラミング言語のルールに対する違反。通常、プログラムの行が実行される前にインタプリタによって検出されます。
プログラミング初心者の場合は、これらのエラーを見逃さないために、頻繁にプログラムを実行してください。

新しいプログラマーによく見られるエラーはタイポです。すべてが正しくスペルされていることと、名前が一致していること（大文字小文字を区別します）を確認してください！

エラーの種類

実行時エラー(Runtime Error) - プログラムがインタープリターによって実行され、その後クラッシュすること。

実行時エラー (Runtime Error) - 正しい構文ですが、プログラムがゼロで割るなど不可能なことを試みたり、文字列を掛け算しようとすること。

プログラムは直ちに停止し、実行中のコード行でエラーを報告するため、しばしばクラッシュと呼ばれます。一般的な実行時エラーの種類：

- 構文エラー (SyntaxError) - 理解できないコード（ただし、起動時に捕捉されなかった）
- インデントエラー (IndentationError) - プログラムの行が正しくインデントされていない
- 値エラー (ValueError) - 無効な値が使用されている（例：int() に文字列値を渡す）
- 名前エラー (NameError) - プログラムが存在しない変数を使用しようとする
- 型エラー (TypeError) - 操作が不正な型を使用する（例：整数を文字列に加算する）

エラーの種類

セマンティックエラー(Semantic Error) – プログラムの意図された目的が達成されていない

セマンティックエラー (Semantic Error) – プログラムは実行されるが、意図した通りの動作をしない。論理エラーとも呼ばれる。

例:

- プログラムは x と y を加算し、その結果を表示するはずなのに、常に x の値を表示する。
- プログラムはファイルからデータを読み込むはずなのに、そのデータを保持している変数は常に空である。

プログラミングのベストプラクティス

他の人がコードを読みやすくするために：

なぜそのようにしたのかを説明するためにコメントを使う

コードのグループを分けるためにホワイトスペース（例えば、空行）を使う

明確にするために質問することを恐れないでください… しかし、まずは自分で理解しようと努力してください。 コードを試してみることを怖がらないでください。