Python の基本事項の学習

このラーニング ラボでは、Python のシンタックス (構文)、スコープ(有効範囲)、変数、演算子、単純な条件分岐の基本事項を学習します。

目標

所要時間:45 分

- Python の基本的なインデントとスコープを理解しよう
- 変数の割り当て方法と使用方法を学習しよう
- Python 割り当て演算子と比較演算子を使用してみよう
- Python 条件分岐を使ってみよう

前提条件

- ページ上部のリンク「**How to Set up Your Own Computer(コンピュータを設定する方法)**」をクリックし、指示にしたがってラボをセットアップしてください。
- サンプル コードを実行するには、使用しているマシンに Python 3 がインストールされている必要があります。

Git Repo を複製する

- **DevNet イベントで DevNet ラーニング ラボ PC を使用している場合は、**
  - タスク バー上の *Git CMD* アイコンをクリックして Git コマンド ウィンドウを開くか、またはスタート ボタンをクリックして、実行バー **git cmd** と入力して **Enter** キーを押してください。
- **自分のワークステーションで作業を行っている場合は、**デスクトップでコマンド ターミナルを開きます。

Linux/Mac OS	Windows
1. **cd /** と入力して、ルート ディレクトリに移動します。 2. **mkdir DevNetCode\<your-name>** と入力して、/DevNetCode/<yourname> というディレクトリを作成します。例:**mkdir –p /DevNetCode/brTiller** 3. **cd /DevNetCode/<your-name>** と入力して、新しいディレクトリに移動します。 例:**cd /DevNetCode/brTiller**	1. **cd ** と入力して、ルート ディレクトリに移動します。 2. **mkdir \DevNetCode\<your-name>** と入力して、\DevNetCode\<yourname> というディレクトリを作成します。例:**mkdir \DevNetCode\brTiller** 3. **cd \DevNetCode\<your-name>** と入力して、新しいディレクトリに移動します。 例:**cd \DevNetCode\brTiller**

- GitHub から devnet-express-code-samples リポジトリを複製します。次のコマンドを入力します。

```
git clone https://github.com/CiscoDevNet/devnet-express-code-samples
```

```
C:\DevNetCode\brTiller>git clone https://github.com/CiscoDevNet/devnet-express-code-samples.git
Cloning into 'devnet-express-code-samples'...
remote: Counting objects: 148, done.
remote: Total 148 (delta 0), reused 0 (delta 0), pack-reused 148Receiving objects:  60% (89/148)
Receiving objects: 100% (148/148), 41.09 KiB | 0 bytes/s, done.
Resolving deltas: 100% (78/78), done.
Checking connectivity... done.
```

作成したディレクトリ内で、Windows の場合は `dir`、Linux/Mac OS の場合は `ls` と入力します。ディレクトリ devnet-express-code-samples が表示されます。

```
C:\DevNetCode\brTiller>dir
 Volume in drive C is System
 Volume Serial Number is 808B-CEEA

 Directory of C:\DevNetCode\brTiller

12/15/2015  03:16 PM    <DIR>          .
12/15/2015  03:16 PM    <DIR>          ..
12/15/2015  03:16 PM    <DIR>          devnet-express-code-samples
               0 File(s)              0 bytes
               3 Dir(s)  55,545,159,680 bytes free
```

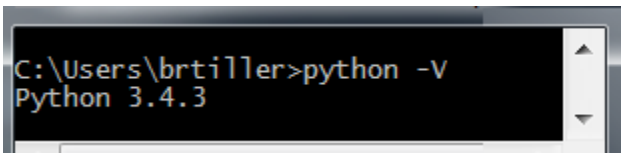
ステップ 1：Python のバージョンと対話型シェル

Python のバージョンを確認

システムにインストールされている Python のバージョンは簡単に確認できます。たとえば、Windows オペレーティング システムの場合は、ターミナル ウィンドウにアクセスし、コマンド プロンプトで `python -V` と入力してリターン キーを押します。お使いのシステムに複数のバージョンの Python または Python 3 がインストールされている場合は、オペレーティング システムに応じて `py -3 -V` または `python3 -V` と入力します。詳細については、下の図を参照してください。

Python バージョン コマンド	オペレーティング システム
python -V または py -3 -V	Windows
python3 -V	Linux、OS X

Python のバージョンを確認する例を示します。

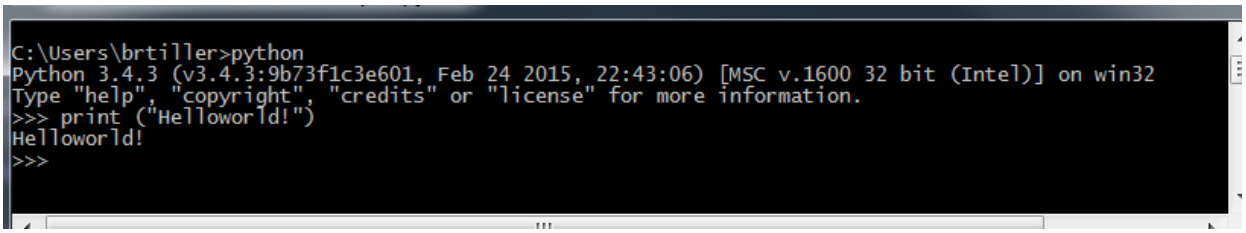


Python 対話型シェルの使用

Windows では、コマンド プロンプトで `python` と入力してリターン キーを押すだけで、Python の対話型シェルを開くことができます。対話型シェルでは、コマンドライン上で Python コードを記述して、対話形式で Pythonのコードを実行していくことができます。終了するには、`quit()` と入力し、`Enter` キーを押します。

Python 対話型シェル	オペレーティング システム
python または py-3	Windows
python3	Linux、OS X
対話型シェルの終了	quit()

次に Python 対話型シェルの使用例を示します。



実際にやってみよう

Python の対話型シェルを使ってみる

1. ターミナルを開きます。
2. 利用しているオペレーティング システムに応じて適切な Python コマンドを入力し、バージョンを確認します。たとえば Windows の場合は、`python -V` と入力してリターン キーを押します。

Python 対話型シェルの使用

1. 適切な Python コマンドを入力して、対話型シェルを開始します。たとえば Windows の場合は、`python` と入力してリターン キーを押します。
2. `print("Hello World! How are you?")` と入力してリターン キーを押します。
3. 操作の終了後、対話型シェルを終了するために `quit()` と入力し、`Enter` キーを押します。

次のステップ：Python スクリプトとその実行方法について学習しましょう

ステップ 2：Python スクリプトとその実行方法

このステップでは、Python スクリプト、およびスクリプトの実行方法について学習しましょう。

Python スクリプトの記述

Python スクリプトはテキストファイルで、基本的な特徴は次の 2 つだけです。

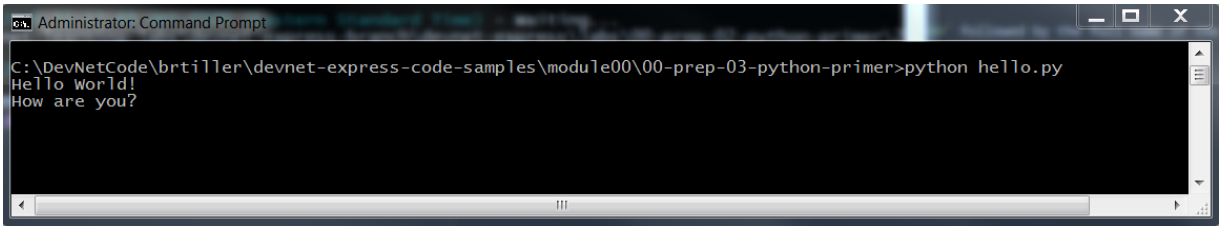
1. スクリプトは、Pythonの構文、文法で決められている書き方で記述していく必要があります。たとえば、ファイルに `print("I'm a Python script!")` と記述して保存します。関数 `print` は Python の組み込み関数で、この書き方は、Pythonの構文、文法に従っているため、このスクリプトファイルは、Pythonのスクリプトとして成り立っています。詳細は別途示します。
2. Python スクリプトのファイルの拡張子は、`.py` にするのが一般的です。たとえば、スクリプト名を `myscript.txt` にした場合、スクリプト名が `.py` で終了しないため、このファイルは Python スクリプトにはなりません。自分がPython スクリプトであると主張するには、ファイル名を `myscript.py` に変更するのがよいでしょう。

Python スクリプトの実行

Python スクリプトは簡単に実行できます。たとえば、Windows オペレーティング システムの場合は、ターミナル ウィンドウのコマンド プロンプトにアクセスし、`python`、Python スクリプトの完全名の順に入力し、リターン キーを押します。Python スクリプトが保存されているディレクトリ以外の場所では、スクリプトのあるディレクトリの場所、続けてスクリプトの名前を正確に指定する必要があります。

Python スクリプトの実行	オペレーティング システム
python script.py または py -3 script.py	Windows
python3	Linux, OS X

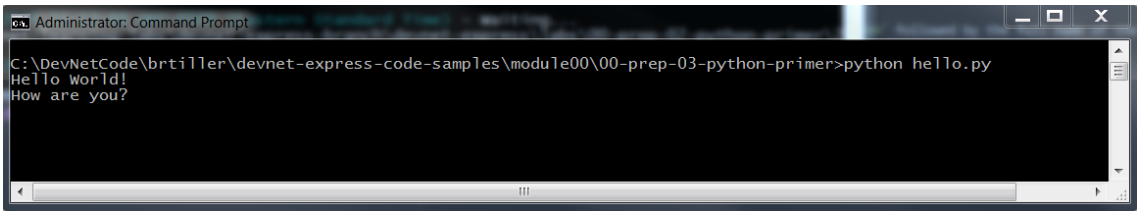
Python スクリプトを実行する例を示します。



実際にやってみよう

Python スクリプトを実行する

1. ターミナルを開き、先に作成したディレクトリ **DevNetCode\<your-name>** に移動します。
    - 「**前提条件**」セクションでは、ソース コード ファイルを Git リポジトリからこのディレクトリに複製しました。作成したディレクトリに **devnet-express-code-samples** サブディレクトリが確認できるはずです。このサブディレクトリがない場合は、前提条件のセクションに戻り、手順に従ってディレクトリを作成し、Git リポジトリを複製します。
2. ディレクトリ **module00\00-prep-03-python-primer** に移動します。ターミナルに、`cd \DevNetCode\<your-name>\devnet-express-code-samples\module00\00-prep-03-python-primer` と入力しましょう。<your-name> はディレクトリに指定した名前です。
3. スクリプトを実行するには、Python コマンドを入力後、コマンド プロンプトにファイル名を入力して、リターン キーを押します。
    - *Windows では*、`python hello.py` または `py -3 hello.py` と入力します。
    - *Mac OS または Linux では*、`python3 hello.py` と入力します。
4. プログラムが実行され、以下のような結果になります。



次のステップ：Python のスコープ、変数、演算子、単純な条件ステートメントについて学習しましょう

ステップ 3：Python のシンタックス(構文)、スコープ(有効範囲)、変数、演算子、単純な条件分岐

このステップでは、Python のシンタックス(構文)、スコープ(有効範囲)、変数、そして条件分岐の基本事項を確認しましょう。

Python の基本

他の言語でコードを記述したことがあれば、スコープ（ステートメントの影響範囲または、変数が存在する有効範囲）を定義する範囲が波括弧 `{}`を使って指定されているのをよく見るでしょう。これらの括弧は、ステートメントの開始と終了、つまりスコープの定義に使用されます。これらの括弧は、ステートメントの開始と終了、つまりスコープの定義に使用されます。

以下に C 言語の例を示します。波括弧が if 文による条件分岐の開始と終了に使用されています。

```
#include <stdio.h>


printf("Hello World!");
num = 1;
if(num < 1){
   printf("This is C language and I'm less than 1!");
   printf("Goodbye Cruel World!");
}
```

Python では、スコープの定義に波括弧を使用せず、代わりにインデントを使用します。インデントの形式としては、スペースまたはタブを使用しますが、どちらかを一貫して使用する必要があります。一貫していないと Python のインタプリタでエラーが発生します。

次の Python コードを見てください。

```
print ("Hello World!")
num = 1
if num < 1:
  print ("I'm less than 1!")
  print ("Goodbye Cruel World!")
```

Python スクリプトを実行すると、Python のインタプリタによってコードが最初から読み込まれ、上から順番に 1 行 1 行処理されていきます。各コードに応じて、処理が実行されるか、今後使用するために情報がメモリ内に保管されます。

1. 最初の行では、Python のインタプリタはテキスト `Hello World!` を画面に出力します。`Print` は Python の組み込み関数です。ここでは、関数に渡されている`Hello World!` という文字列を画面に表示するよう指示しています。
2. 2 行目では、**num** という変数を作成し、値として 1 を割り当てています。この変数の名前は「brett」でも「a」でも問題ありません。詳細については、変数の名前付けに関するページ [英語] を参照してください。Python 対話型インタプリは、変数 **num** の値が 1 であるという情報を保存します。
3. 次の行では、Python 対話型シェルは変数 **num** の値が 1 未満かどうかを確認します。演算子には「`<`」(より小さい)を使用します。Python には多様な演算子が用意されていますが、最も一般的に使用するのは比較、割り当て、算術 [英語] 演算子です。この if 文 [英語] は、条件を確認するための *条件文*と呼ばれます。条件が真の場合、条件文の下のインデントで区別されたスコープ内部のすべてのコードが実行されます。条件が偽の場合、条件文の下のインデントで区別されたスコープ内部のすべてのコードがスキップされます。この場合は、変数 **num** の値が 1 で、1 未満ではないため、条件文は偽になります。シンタックスに関しては、`if 文`はコロンで終了します。Python では、条件文、関数、およびループなどの多くはコロンで終了します。詳細は別途示します。
4. if 文の下の 2 行はインデントされています。このインデントは、これらのコードが条件分岐のスコープ内にあることを示しています。この条件文は偽であるため、この条件分岐のスコープの範囲内にあるこれら 2 つのコードはスキップされます。

クイズ:解答は下を参照

1. 上記の Python コードを実行すると、画面に何が出力されるでしょうか。それはなぜでしょうか。
2. print("Goodbye Cruel World!") の行をインデントしない場合、画面に何が出力されるでしょうか。それはなぜでしょうか。
3. **num** の値を 0 に変更すると、画面に何が出力されるでしょうか。それはなぜでしょうか。

解答

1. 画面には `Hello World!` が出力されます。最初のprint関数呼び出しには、条件分岐はないため、必ず実行されます。その後の条件文`if num < 1` は偽なので、そのスコープ内のコードはスキップされるため、スコープ内のprint関数の呼び出しは実行されません。
2. `Hello World! Goodbye Cruel World!` が出力されます。Hello World! は、条件分岐がないため、必ず表示されます。Goodbye Cruel World! も同じ理由で出力されます。
3. `Hello World! I'm less than 1! Goodbye Cruel World!` すべてが出力されます。Hello World! は、条件分岐がないため、必ず表示されます。条件文 `if num < 1` が真になるので、この条件分岐のインデントで区別されたスコープ内に含まれるコードも実行されるようになります。

実際にやってみよう

1. ターミナルを開き、ステップ 1 で作成した **DevNetCode\<your-name>** という名前のディレクトリに移動します。
2. ディレクトリ **module00\00-prep-03-python-primer** に移動します。ターミナルに、`cd \DevNetCode\<your-name>\devnet-express-code-samples\module00\00-prep-03-python-primer` と入力します。<your-name> はディレクトリに指定した名前です。
3. スクリプトを実行するには、Python コマンドを入力後、コマンド プロンプトにファイル名を入力して、リターン キーを押します。

    - *Windows では* **py -3 helloworld.py** と入力します。または **python helloworld.py** と入力します。
    - *Mac OS または Linux では* **python3 helloworld.py** と入力します。
4. 画面に表示される結果を確認しましょう。

以下の変更を行い、helloworld.py スクリプトを実行しましょう。

1. **helloworld.py** ファイルを開きます。たとえば Windows では、**notepad helloworld.py** と入力します。
2. クイズの問題 2 で示された変更を行いましょう。
3. ファイルを保存します。エンコーディング タイプがオプションの場合は、**UTF-8** を選択します。
4. スクリプトを実行しましょう。
5. ステップ 1 ～ 4 を繰り返し、クイズの問題 3 で示された変更を行いましょう。

次のステップ：Python の演算子と条件分岐についてもう少し詳しく学習しましょう。

ステップ 4：Python の演算子と条件分岐

このステップでは、Python の演算子と条件分岐について、もう少し詳しく学習しよう。

Python の演算子

Python には多くの演算子が用意されています。詳細については Python の基本演算子に関するページ [英語] を参照してください。ここでは、基本である代入演算子と比較演算子に焦点を当てます。

代入演算子

代入演算子は等号記号（`=`）で、変数に値を代入するために使用されます。値を割り当てて、記憶しておくことができます。記憶されているので、代入された変数を使って、あとで、代入した値にアクセスすることができます。これまでの演習の中で登場した `num = 1` というコードに見覚えがあるでしょう。この代入は、新たに作成された変数 **num** に値 1 を記憶しておいてちょうだいねということを意味します。この変数にはその後、別の値を代入することもできます。別の値を代入すると、記憶されている内容は書き換えられていき、最後に代入した値が記憶されていることになります。ここで `num = 2` を代入すると、前の値が上書きされ、**num** の値は 2 になります。

これ以外にも、演算子を組み合わせた代入演算子も利用できます。たとえば、`num +=1` では値を追加して結果を代入します。このコードは `num = num + 1` と同等のコードであるため **num** の値が 1 の場合は 2(1 + 1 = 2) になります。ただしこの演習では、ひとまず、単純な代入である = を覚えておきましょう。

比較演算子

2 つの値を比較したときの結果は、条件が満たされて真になるか、条件が満たされずに偽になるかの 2 つのみです。これらの結果に基づいて、特定のアクションが実行されかどうかが決まります。次に示す比較演算子について考えてみてください。この例では、変数 a=1、変数 b=2 と仮定します。

演算子	説明	例
==	2 つのオペランドが等しい場合、条件は真になる。	(a == b) は真ではない。
!=	2 つのオペランドが等しくない場合、条件は真になる。	(a != b) は真である。
<>	2 つのオペランドが等しくない場合、条件は真になる。	(a <> b) は真である。 != 演算子と似ている。
>	左側のオペランドの値が右側のオペランドの値より大きい場合、条件は真になる。	(a > b) は真ではない。
<	左側のオペランドの値が右側のオペランドの値より小さい場合、条件は真になる。	(a < b) は真である。
>=	左側のオペランドの値が右側のオペランドの値より大きいまたは等しい場合、条件は真になる。	(a >= b) は真ではない。
<=	左側のオペランドの値が右側のオペランドの値より小さいまたは等しい場合、条件は真になる。	(a <= b) は真である。

ステップ 3 で使用した以下の Python コードについて考えてみましょう。比較を含む条件文 `if num < 1:` では、小なり演算子 `<` が使用されており、この条件は偽になります。この条件はが偽になる理由はお分かりだと思います。変数 **num** には 1 が割り当てられており、1 は 1 未満ではないためです。

```
print ("Hello World!")
num = 1
if num < 1:
  print ("I'm less than 1!")
  print ("Goodbye Cruel World!")
```

クイズ：解答は下を参照

1. 上記の Python コードでは、`<` の代わりにどの比較演算子を使用すると、条件文 `if num <ここに演算子を挿入> 1` が真になりますか。該当するすべての演算子を挙げ、その理由を説明してください。
2. 変数 **num** に値 0 が割り当てられている場合、どの比較演算子を使用すると条件文 `if num <ここに演算子を挿入> 1` が偽になりますか。該当するすべての演算子を挙げ、その理由を説明してください。

解答

1. `==`、`>=`、`<=`。1 は 1 に等しいので演算子 == は真です。1 は 1 以上なので演算子 >= は真です。1 は 1 以下なので演算子 <= は真です。
2. `==`、`>`、`>=`。0 は 1 に等しくないので演算子 == は偽です。0 は 1 より大きくないので演算子 > は偽です。0 は 1 以上でないので演算子 >= は偽です。

Python 条件ステートメント

これまで説明した例のコードでは、1 つの条件文 `if num < 1` を見てきました。最初の例では、変数 **num** が 1 に等しく、1 は 1 未満ではないため、この条件文は偽になります。この評価の結果、この条件分岐のインデントで区別されたスコープのコードはスキップされます。さて、ある条件を満たした際にはある処理を行い、その条件を満たさなかった場合に、さらに条件を確認して、処理をしたい場合もあるでしょう。そして、すべての条件を満たさなかった場合に、デフォルトの処理を行いたい場合もあるでしょう。次の Python コードを見てください。

```
print ("Hello World!")
num = 1
if num < 1:
 print ("I'm less than 1!")
 print ("Goodbye Cruel World!")
elif num == 1:
  print("I'm equal to 1!")
else:
  print("I'm greater than 1!")
```

上記のソース コードには、最初の条件を満たさなかった場合に、さらにほかの条件を評価するための処理と、すべての条件がまったく満たされない場合のデフォルト アクションが追加されています。これらの条件分岐では、真と評価された最初の if 文のスコープ内のコードが実行され、残りの条件文内のスコープの範囲内のコードはスキップされます。たとえば、このケースでは変数 **num** は 1 に等しく、最初の条件文 `if num < 1` は偽になるため、次の文 `elif num ==1` が評価されます。この条件文は真になるため、`print("I'm equal to 1!")` が実行されます。

変数 **num** に値 2 が割り当てられても、同じ評価パスをたどります。最初の条件文 `if num < 1` は偽になるため、次の条件文 `elif num ==1` が評価されますが、これも偽になります。条件がまったく満たされないため、最後の else 内のインデントで区別されたスコープ内のコードである `print("I'm greater than 1!")` が実行されます。

実際にやってみよう

初めてのプログラムを作成する

1. ターミナルを開き、ステップ 1 で作成した **DevNetCode\<your-name>** という名前のディレクトリに移動します。
2. ディレクトリ **module00\00-prep-03-python-primer** に移動します。ターミナルに、`cd \DevNetCode\<your-name>\devnet-express-code-samples\module00\00-prep-03-python-primer` と入力します。<your-name> はディレクトリに指定した名前です。
3. `myworld.py` というファイルを作成しましょう。
4. 次のことを実行する Python コードを記述しましょう。
   a. 変数に値を割り当てる
   b. 変数の値によって条件分岐してみよう(if 文の書き方を思い出そう)
   c. 条件が真の場合のみ、何か画面に文字列を表示してみよう(if 文が有効なスコープはインデントで区別されている必要がある点に注意しよう)
5. ファイルを保存し、スクリプトを実行し、コードを確認してみよう。
6. 記述したコードの条件分岐に、さらに別の条件文と、すべての条件を満たさなかった時のデフォルトの処理を追加し、それぞれの条件を満たした際に、それぞれ別の文字列を画面に表示するようにしてみよう。実行して結果を確認しましょう。
7. 作業が完了したら、自分で記述したコードとファイル `myworldsol.py` を比較しましょう。

次のステップ：Python の文字列連結について学習します。

ステップ 5：Python の文字列連結

このステップでは、Python の文字列連結方法を学習します。

Python の文字列の定義

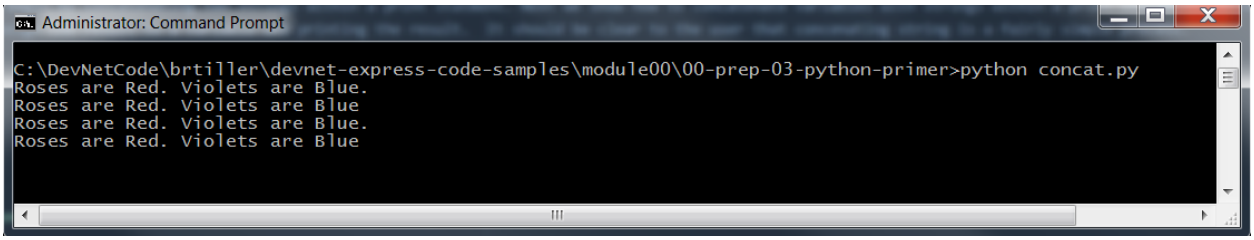
これまでのステップで、すでに Python 文字列を引用符で囲み英数字データとして定義しました。引用符は、一重引用符 `''` と二重引用符 `""` のどちらも使用できます。文字列には、エンコーディングに必要なだけ任意の文字を含めることができます。ただし、この演習では目的どおり基本操作を学習します。

+ 演算子を使用した Python 文字列の連結

エンジニアは、読みやすさや出力を考慮して、多くの場合 Python 文字列やその他のデータ型を連結して変数に割り当てたり、画面に表示したりします。Python での文字列の連結 [英語] 方法はいくつか存在しますが、ここではまずプラス（+）演算子を使用する最も簡単な方法を説明します。いくつか例を見てみましょう。

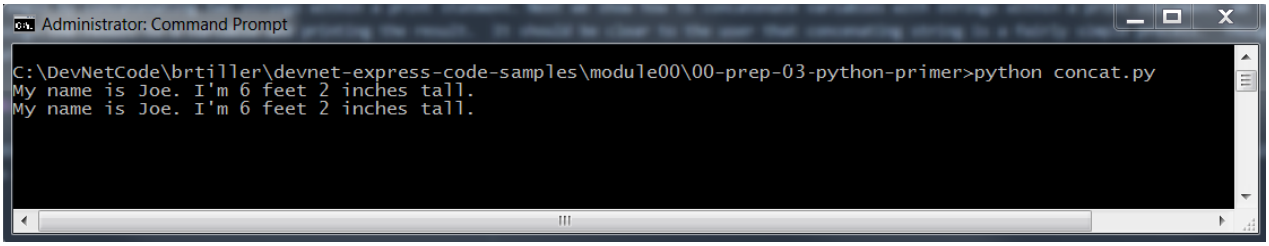
```
myVarRed= "Red"
myVarBlue= "Blue"


print("Roses are Red." + "Violets are Blue.")
print("Roses are " + myVarRed +".Violets are " + myVarBlue)


myStr ="Roses are Red." + "Violets are Blue."
varStr = "Roses are " + myVarRed +".Violets are " + myVarBlue


print(myStr)
print(varStr)
```

上記の例では、まず print ステートメント内で 2 つの文字列を連結しています。次に print ステートメント内で変数と文字列を連結しています。最後に、文字列を連結して、その結果を変数に割り当て、出力するという少し複雑なタスクを追加しています。このように、文字列の連結はシンプルなプロセスです。さまざまな文字列や変数を使用してデータを出力していますが、出力結果は次のようになります。



では、異なるデータ型を連結してみましょう。Python は、連結するデータは文字列データ型であると想定していますが、次の例では文字列と整数を使用します。そのため、`str()` という Python の組み込み関数を使用して各整数をラップする必要があります。この関数は、整数のようなシンプルなデータ型を文字列に変換します。

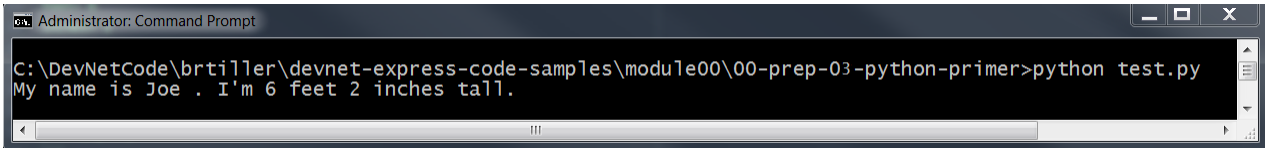
```
name = "Joe"

feet= 6

inches= 2


print("My name is " + name +".I'm " + str(feet) + " feet " + str(inches) + " inches tall.")


myStr = "My name is " + name +".I'm " + str(feet) + " feet " + str(inches) + " inches tall."

print(myStr)
```



演算子を使用した Python 文字列の連結

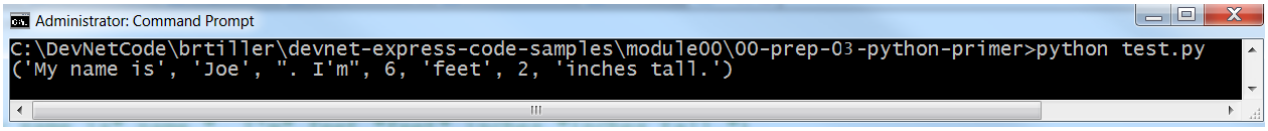
`+` 演算子を使用すると直感的に Python 文字列を連結できますが、例で紹介したように、文字列以外のデータ型を使用して文字列に変換する必要性も生じます。Python では、カンマ(`,`)演算子を使用すると、**演算が print 関数内で行われる場合**、文字列を連結し、この変換を自動的に行うことができます。この新しい演算子を使用する例の最初の数行を確認してみましょう。

```
name = "Joe"

feet= 6

inches= 2


print("My name is ",name,".I'm ",feet," feet ",inches," inches tall.")
```



ただしこの連結演算は、変数に割り当てた後に出力しようとすると、データは連結されていないため期待通りに動作しません。

```
name = "Joe"

feet= 6

inches= 2


myStr="My name is ",name,".I'm ",feet," feet ",inches," inches tall."

print(myStr)
```



実際にやってみよう

1. ターミナルを開き、ステップ 1 で作成した **DevNetCode\<your-name>** という名前のディレクトリに移動します。
2. ディレクトリ **module00\00-prep-03-python-primer** に移動します。ターミナルに、`cd \DevNetCode\<your-name>\devnet-express-code-samples\module00\00-prep-03-python-primer` と入力します。<your-name> はディレクトリに指定した名前です。

Python スクリプト concat.py の実行と編集

1. スクリプト `concat.py` を実行してみましょう。
2. このファイルを開き、`str(feet)` を `feet` に変更しましょう。スクリプトを保存して実行してみましょう。整数は暗示的に文字列に変換できないという型エラーが発生することでしょう。
3. この問題を修正してこのスクリプトを保存し、もう一度実行しましょう。
4. スクリプトに文字列とさまざまなデータ型を連結するコードをいろいろ書いて遊んでみよう。
5. この操作が完了したら、自分で記述したコードと Python スクリプト `concat_sol.py` を比較してみよう。

おめでとうございます。Python 初級レベル 1 のラボを修了しました。