Getting Started with Python

Overview

In this lab, you'll start writing some simple Python script. You'll also create and run some simple code in a Python module.

Source folders

Student folder : `C:\PythonDev\Student\01-GettingStarted`

Solution folder: `C:\PythonDev\Solutions\01-GettingStarted`

Roadmap

There are 3 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Writing Python script interactively
2. Writing Python modules
3. Additional suggestions (if time permits)

Exercise 1: Writing Python script interactively

Open a command window and go to the *student* folder for this lab. In the Python interpreter, create variables to hold some data, e.g. your name and age (hard-coded for now). For example:

```
name = "John Smith"
age = 21
```

Print these variables on separate lines, using the standard `print()` function (note the parentheses are required in Python 3.x, to call the `print()` function):

```
print(name)
print(21)
```

Now try printing these variables in a single string, such as the following:

```
print(name + "is " + age + " years old")
```

You'll get a Type Error from the interpreter now, saying it can't convert the int to a str implicitly. This tells us two things: Python has an int data type, and Python doesn't automatically convert the int to a string. To rectify the problem, use the standard `str()` function as follows:

```
print(name + "is " + str(age) + " years old")
```

Now try the following code. The print() function allows you to specify a format string, containing format specifiers such as %s for string, %d for decimal integer, etc. After the format string, the `% (name,age)` syntax passes all the values required to fill in the blanks. We'll describe this syntax in more detail later:

```
print("%s is %d years old" % (name, age))
```

Here's a full list of format specifiers you can use in the `print()` function. Try these out, to see what effect you get:

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

You can also use escape sequences such as the following… try these out too:

- `\a` audible alert
- `\b` backspace
- `\n` newline
- `\r` carriage return
- `\s` space
- `\t` tab
- `\v` vertical tab

Exercise 2: Writing Python modules

Create a new file and save it as `userinfo.py`. Add code to perform the following tasks:

- Ask the user to enter their name and age.
- Calculate the user's age next birthday.
- Output these details.

Here are some hints:

- To get input from the standard input device (i.e. the keyboard), use the `input()` function. The `input()` function takes a string parameter that displays a prompt message to the user. The function always returns a string.
- When you get the user's age, it'll come back into your program as a string. You'll need to convert it into an int, so you can do maths on it. To do this, call the `int()` function.
- Add some comments to your code, to describe the salient points. Comments start with `#` in Python.
- Run your code and make sure it works.

Exercise 3: Additional suggestions (if time permits)

- Python lets you create multi-line strings, by starting and ending the strings with `"""` (i.e. three quote marks). Experiment with this, to create a formatted message such as the following:

```
=========================
= FORMATTED INFORMATION =
=========================
  Name: Andy
  You'll be 22 soon!
=========================
```

- Experiment with string concatenation. Adjacent string literals are automatically concatenated. Also, you can use the + operator to concatenate variables and string literals.