

Introduction to Programming

Python Lab 5: Strings and Output

Getting Started

- Create a new folder in your disk space with the name **PythonLab5**
- Launch the Python Integrated Development Environment (IDLE) - begin with the **Start** icon in the lower left corner of the screen.
- If you are in a DCSIS laboratory, search using the keyword **Python** and click on **IDLE (Python 3.6 64-bit)**

A window with the title **Python 3.6.2** should appear. This window is the *Shell*.

Getting Started (2)

- If you are in the ITS laboratory MAL 109, then right mouse click on the **Start** icon in the lower left corner of the screen.

A list of menu options should appear and click on **Search**. Type **Python** in the search text box at the bottom of the pop-up window. A list of Apps should appear and select

Python 3.4 IDLE(PythonGUI)

A window with the title **Python 3.4.3 Shell** should appear. This window is the **Shell**.

- In the **Shell** click on **File**. A drop down menu will appear. Click on **New File**. A window with the `title` **Untitled** should appear. This window is the **Editor**.

Getting Started (3)

- In the *Editor*, click on **File**, and then in the drop down menu click on **Save As...** .

A window showing a list of folders should appear.

- To search any folder on the list, double click on the folder.
- Find the folder **PythonLab5** and double click on it.
- In the box **File name** at the bottom of the window
 1. Type `Monogram.py`
 2. Then click on the button **Save** in the lower right corner of the window.

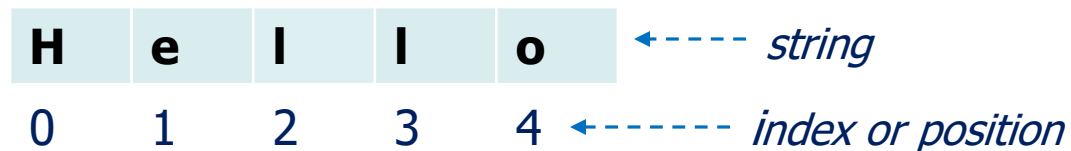
The title of the *Editor* should change to show the location of the file `Monogram.py`

Program `Monogram.py` prints out a monogram

- Learning objective:** Understand how to access an individual character from a string. A string is a sequence of characters.
 - The individual characters in a string can be extracted using an index.

For example, `"Hello"[0]` is the string "H" that contains the first letter of the string "Hello". The index is **0** in this case.

Similarly, `"Hello"[1]` is "e", etc.



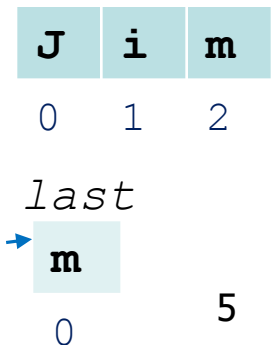
Another example, if the variable `name` is defined as *name*

```
name = "Jim"
```

```
last = name[2]
```

```
# the character at position 2 is stored last
```

```
# as the string "m" in the variable last
```



Program `Monogram.py` prints out a monogram (2)


- **Question 2: Problem statement**


Write a program that prompts a user to enter his or her first name and then prompts the user a second time to enter his or her family name.

The program then creates a string with two characters, consisting of the first character in the first name followed by the first character in the family name. This string is referred to as a **monogram**.

Print out the monogram together with a short statement of the fact that it is a monogram. See PFE R2.14

Program `Monogram.py` prints out a monogram (3)

- **Problem solving** - Convert the following pseudo code into a sequence of Python statements in your program.
 1. Read in the user's first name and store the input string in the variable `first` *
 2. Read in the user's family name and store the string in the variable `surname` *
 3. Create a string with **two characters**, consisting of the first character in the first name followed by the first character in the family name. **Hint:** use the `+` operator to concatenate strings, that is, `first[?] + surname[?]` replace the `?` with a suitable index for the first character of each string. 
 4. Print out the monogram (that is, the string created in step 3 above), together with a short statement of the fact that it is a monogram.

 **Note** *: Use the function `input` to read in the user's first name and family name, e.g. `first = input("Enter your first name: ")`

Program `Monogram.py` prints out a monogram (4)

- Provide a comment at the beginning of the program to explain the purpose of the program along with your name and the date.
- Save the program to the file `Monogram.py` and then run it.

Program `LargeLetters.py`: Writing Large Letters

- Create a new *Editor* for a new file called `LargeLetters.py`

- **Question 3: Problem statement**

A large letter H can be produced like this.

Field width for five characters

*				*
*				*
*	*	*	*	*
*				*
*				*

The grid is included to ensure that the asterisks are in the right places on this page. It is **not** necessary for the program to create or print out a grid. Note that the print out of a string literal can be forced to a new line using the escape character `\n`. For example,

```
print("* * \n * *")
```

prints out the first two rows of the grid in the form

```
* *
* *
```

Program `LargeLetters.py`: Writing Large Letters (2)

- **Problem statement (continued)**

- In your program create a string literal `LETTER_H` such that the instruction

```
print(LETTER_H)
```

produces a large letter **H**.

- Create large versions of the letters **E**, **L**, **O** and write the message **HELLO** in large letters as follows,

H
E
L
L
O

See PFE, P2.18.

Program `LargeLetters.py`: Writing Large Letters (3)

- **Problem solving** - Convert the following pseudo code into a sequence of Python statements in your program.

1. Create a string literal `LETTER_H` as follows



```
LETTER_H = "*" * \n* * \n***** \n* * \n* "
```

2. Create a string literal `LETTER_E` in a similar way as step 1 above.
3. Create a string literal `LETTER_L` in a similar way as step 1.
4. Create a string literal `LETTER_O` in a similar way as step 1.
5. Print out each string literal, in the order shown below:



```
print(LETTER_H)  
print(LETTER_E)  
print(LETTER_L)  
print(LETTER_L)  
print(LETTER_O)
```

Program `LargeLetters.py`: Writing Large Letters (4)

- Provide a comment at the beginning of the program to explain the purpose of the program together with your name and the date.
- Save the program to the file `LargeLetters.py`
- Run your program.

Program `AlignedNumbers.py`: Aligned Numbers

- Create a new *Editor* for a new file called `AlignedNumbers.py`
- **Learning objective:** Understand how to format output using the **String format** operator.
 - Consider the following statements,

```
x = 78  
print("A useful number: %5d" % x)
```

When they are executed the output in the *Shell* window is
A useful number:

			7	8
--	--	--	---	---

- The **format specifier** `%5d` creates a field with five characters. The number 78 is placed in the field and **right justified**. The remaining entries in the field are spaces. There is an additional space in the string "A useful number: ". Thus there is a total of **four** spaces between the colon and the number 78 in the print out. If a field of 6 characters is required then the format specifier `%6d` is used, etc.



Note: use **d** with an integer.

Program `AlignedNumbers.py`: Aligned Numbers (2)



- Recall that the `%` symbol is used to find the remainder of a floor division. However, that is only the case when the values on the left and right of the remainder operator are both numbers.
- If the value on the left of the `%` symbol is a string, in this case `"A useful number: %5d"`, then the `%` symbol becomes the **string format** operator.
- The letter `d` at the end of the format specifier shows that we are formatting an integer value.

Program `AlignedNumbers.py`: Aligned Numbers (3)



- **Question 4: Problem statement**

Write a program that prompts the user for **two non-negative integers**, calculates the sum s and the product p of the two integers and then displays s and p right justified. For example, if the sum is 22 and the product is 121, then the display is

```
Sum:           22
Product:       121
```

Choose the two format specifiers such that s and p are aligned, provided the two integers which are input each have at most three digits. See PFE P2.5.

Program `AlignedNumbers.py`: Aligned Numbers (4)

- **Problem solving** - Convert the following pseudo code into a sequence of Python statements in your program.

1. Read in the first non-negative integer, as follows



```
a = int(input("Enter a non-negative integer: "))
```

2. Read in the second non-negative integer in a similar way as step 1, and store the user input in the variable `b`.
3. Calculate the sum `s` of the two integers and then print out `s` right justified.
4. Calculate the product `p` of the two integers and then print out `p` right justified.

Program `AlignedNumbers.py`: Aligned Numbers (5)



- **Think about:** What happens if `s` or `p` is larger than expected?
- Provide a comment at the beginning of the program to explain the purpose of the program together with your name and the date.
- Save the program to the file `AlignedNumbers.py`
- Run your program.

Supplementary Questions for Private Study

- The laboratory worksheet contains supplementary questions in section 5 for private study.
- You are encouraged to complete the supplementary questions at home, or in the laboratory if you have time after completing questions 2 to 4.