LLP109 - Digital Application Development

Introduction to python™

Dr. Hyun Lim

h.lim@lboro.ac.uk

Institute for Digital Technologies

Loughborough University London

Disclaimers

I have tested all the code examples included in the lecture notes in Python 3.6 environment. However, these notes may contain trivial/ factual errors as the syntax slightly differs according to your version of Python, e.g. Ver. 2.7 and 3.7.

Time passes, everything changes. It is natural that any (computer) languages keep changing.

Reading List

Books (ebook/hardcopy)

- ✓ **Python programming** : an introduction to computer science, John M. Zelle
- ✓ **Learning Python** : learn to code like a professional with Python, Fabrizio Romano
- ✓ **Learning Python**, Mark Lutz

Some other books, including **ebooks,** are also available in the **main**/ **London** Library. Have a search of the library catalogue: https://vufind.lboro.ac.uk/

Excellent Free e-Books

- Allen Downey, Jeff Elkner and Chris Meyers, **Learning with Python** How to Think Like a Computer Scientist
  https://greenteapress.com/wp/learning-with-python/

- C.H. Swaroop, **A Byte of Python**
  https://python.swaroopch.com/

Various module-specific resources are available from the library and online.

Contents

- Introduction
- Data Types/Structure
- Control flow

First Computer Programmer

Ada Lovelace (1815-1852)

Lord Byron's daughter, an English mathematician and writer, proposed mechanical general-purpose computer, the Analytical Engine. She was the first to recognise that the **machine can be applications beyond pure calculation**, and published the **first algorithm**.

- Wikipedia

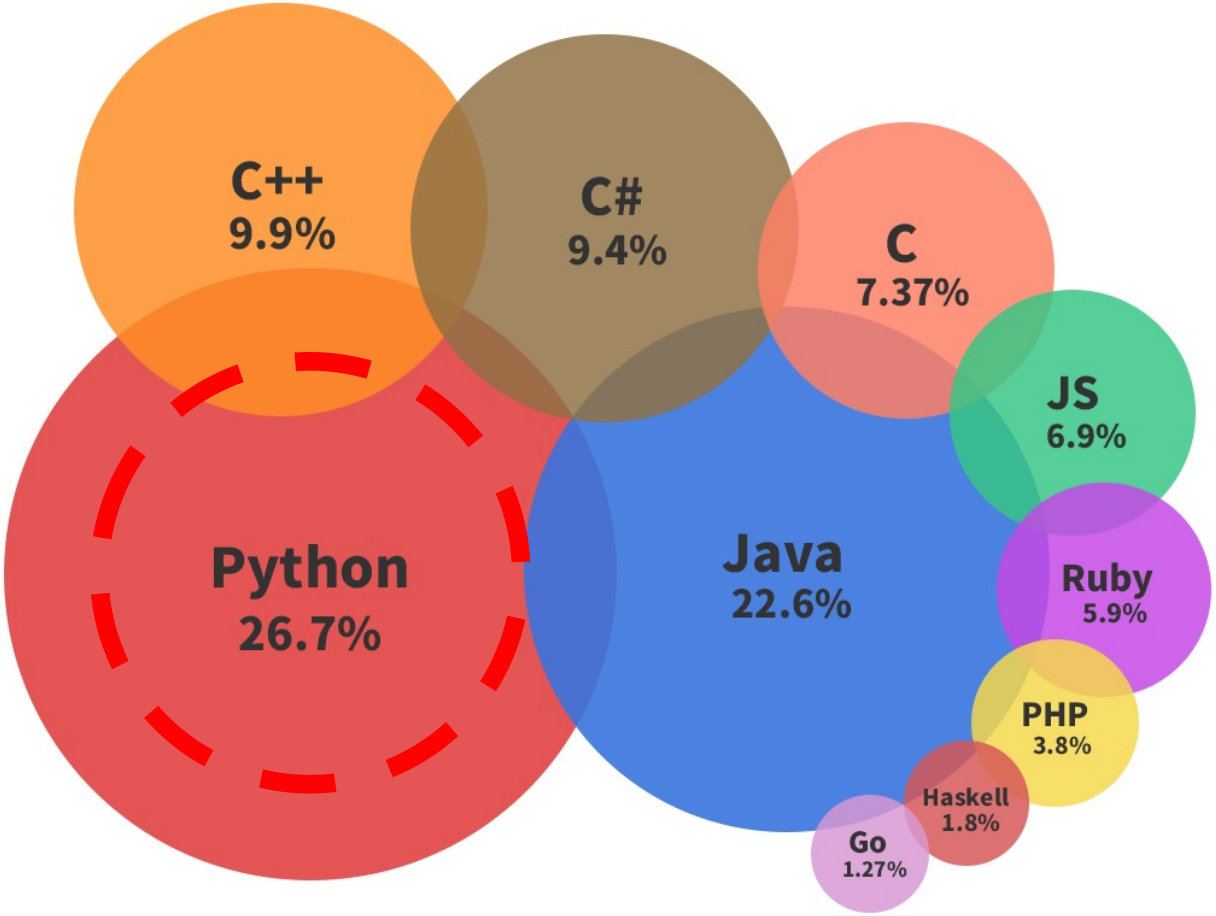
python™

has been published in 1991

- C++ (1979)
- **Python (1991)**
- JAVA (1991)

"Monty **Python**'s Flying Circus",
a BBC comedy series (1970s)

Popular Programming Languages



Development Environments (editors and compilers)

1. **IDLE** (default if you have installed Python)

   Download and install ***Python 3*** in your computer at

   softwarecenter:SoftwareID=ScopeId_369856DC-9EA9-48E8-8227-77CCDB1E4E48/Application_019d72ab-618f-42ea-a61b-8b45e348d795

   Or https://www.python.org/downloads/

2. **PyCharm** (Community: free):

   https://www.jetbrains.com/pycharm/download/#section=windows

3. **Jupyter:** https://jupyter.org/install

4. **Visual Studio**

   Install additional software modules including Python
   Visual Studio installer > ***More/Modify*** > [V] ***Python***

Let's agree that you do **NOT ask me how to use your compiler tools after today**, especially during the lectures as it is not within the scope of the lectures.

Useful Websites

Documentation, tutorials, beginner's guide, discussion, Q&A

- ✓ http://python.org/
- ✓ http://wiki.python.org/moin/PythonBooks

Summary

- Course overview, including aims, objectives, ILO, assessment (coursework), off-/online resources

Let's think & interpret things **logically** in coding !

Python †IDLE Shell

- *Interactive mode (Interpreter prompt)*

Run Python **IDLE** > just write your code **at the prompt** and press the ***Enter*** key

```
>>> print ("Hello World!")
Hello World!
>>> 1+2*3
7
>>> name = "Hyun"
>>> name
Hyun
>>> print ("Hello", name)
Hello Hyun
>>>
```

† IDLE (Integrated Development and Learning Environment)

Compiler vs *Interactive* mode

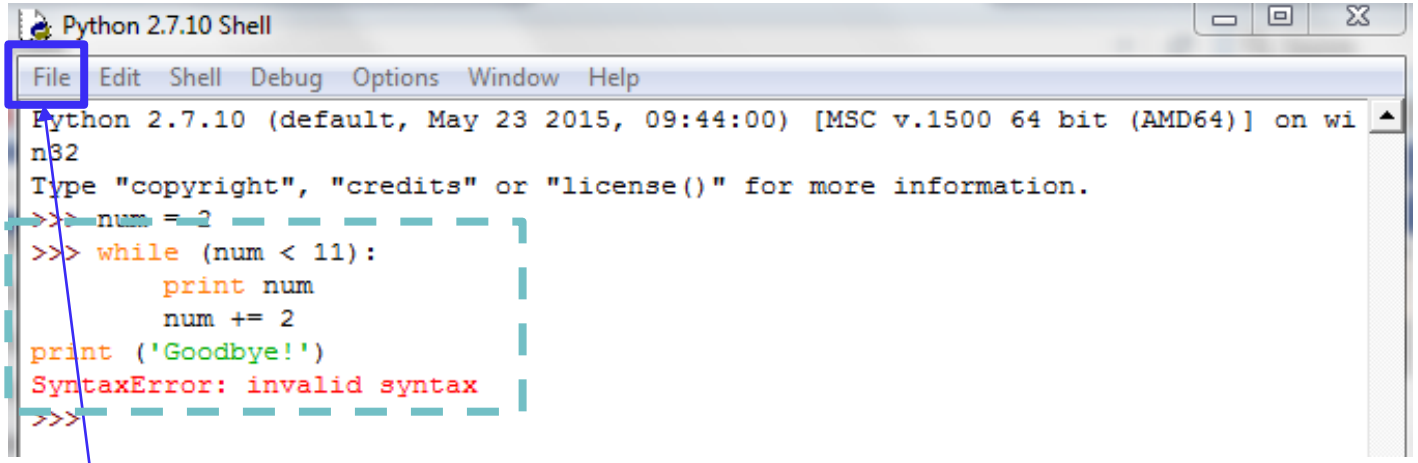
- The **interactive** mode in Python takes **one statement at a time** and executes the instructions.

- The **compiler** converts high-level programming language (of **the whole program**), i.e. Python, to low-level language ***at once***, i.e. ***machine language*** that is to execute the program.

IDLE Shell

- **Interactive mode**: to run only **a single line** of your code, use a commend prompt window:

```
>>> 1+2
3
>>>
```



- **Compiler mode** (Source Code Encoding): to run a program including **multiple lines:**

  (at the top menu) **File** > **New file** > (write your code in the new edit window) > **Save as** (in the File menu) > xxxx.py

  then, run the whole program by choosing **Run** > **Run module** (top menu) or just pressing **F5** key

Comments in Python

```
>>> print ("Hello World!")  # This is my first code in Python.
"""
Triple quotes (") will work just like a block comment, such as,
1+2*3
print "Hello again"
"""
```

- A ***comment*** starts with a **#**, then Python will ignore them.
- In-code documentation
- There is no block comment in Python. However, triple quotes (""" """or ''' ''') will work just like a block comment.
- In Visual Studio (only)

  Comment selection: ***ctrl+k*** & ***ctrl+c***, Uncomment: ***ctrl+k*** & ***ctrl+u***

Newline, tab, backslash

\n : newline

\t : tab

**** : \ (backslash)

```
>>> print ("Hello\t\tWorld!", "\n\n\n")
Hello        World!
        #  2 tabs (empty space)
                                    #  3 new (empty) lines


>>>
```

Variables – assignment, delete

```
>>>x = 4          # assignment operator (=), x is of type int
>>>y = "Sally"    # y is of type str
>>>print(x)       # Or, type just x and press Enter, instead
# assign values to multiple variables in one line:
>>>f0,f1,f2="Orange","Banana","Cherry"
>>>f0
'Orange'
#  A variable name cannot start with a number, e.g. 1x, 2_item

# By the way, this is a comment
# A computer does not recognise whatever you write after #

>>> del x  # empty the variable
# ctrl + F6 or [Restart Shell] in the Shell menu removes all the
variables defined in IDLE.
```

Variable Types - *Numbers*

```
>>>x=1        # int
>>>y=2.8      # float
>>>z=1+3j     # complex
```

Type Casting

```
>>>a=float(x)        #convert from int to float:
>>>b=int(y)          #convert from float to int:
>>>c=complex(x)      #convert from int to complex:
>>>d=str(y)          # d is a string '2.8'. d is not a number
```

>>>~~d+2~~ # d is **not a number**. '2.8'+2 is **not** possible

TypeError: cannot concatenate 'str' and 'int' objects

Summary

- Python development environments: **IDLE**
- Have created **your first Python code**: *"Hello World!"*
- Variables: *str*, *int*, *flot*
- *# Comment*

Contents

- Introduction
- Data Types/Structure
- Control flow

Composite data type

Data Structures - *String*, *Tuple*, *List*

immutable **mutable**

(Character/text) ***String***
- immutable

Hello

(index)	0	1	2	3	4
	-5	-4	-3	-2	-1

```
>>> abc="abcdefgi"   # It is a string. ' ' is also fine, instead of " "
>>> abc[0]           # [ ] calls elements of a string
 ?
>>> abc[1]
 ?
>>> abc[-1]          #  A[-1] returns the last item
 ?
>>> abc[1:5]         #  A[n:m] returns items from n to m-1
  ?
>>> abc[5:-1]        # i.e. from n (inclusive) to m (exclusive)
 ?
>>> abc[1] ='k'      # immutable
TypeError: 'str' object does not support item assignment
```

How to update part of a ***string*** array

```
>>> my_email = "jamie@lboro.ac.uk"
>>> my_email = "jim" + my_email[5:]
>>> my_email
'jim@lboro.ac.uk'
```

jamie@lboro.ac.uk

[n:] means all the items from the element *n*.

Quiz

```
>>> my_email = "john" + my_email[- ? :]
>>> my_email
'john@lboro.ac.uk'
```

-? -2 -1

```
>>> my_email[:5]   #  ?
```

print (……)

% : a print format operator

```
>>> "%s lives in %s at latitude %f" % ("John", "London", 51.50)
'John lives in London at latitude 51.500000'
```

Tuple - **immutable** sequence

```
>>> a_tp = (1, 2, 3)          # or just, a_tp = 1,2,3
>>> a_tp[0]
?
>>> a_tp[1:]
?
>>> a_tp[0] = 5               # immutable
Traceback (most recent call last):
  File "<pyshell#189>", line 1, in <module>
    a_tp[0] =5
TypeError: 'tuple' object does not support item assignment

>>> a_str = '12345'           # a string
>>> b_tp = tuple(a_str)       # type casting a string to a tuple
('1', '2', '3', '4', '5')
```

Tuple - Complex data type

```
names_tp = ( 'Tom', 'James', 'Chen')
# or just, names_tp = 'Tom', 'James', 'Chen'

b_tp = ('1', '2', '3', '4', '5')
c_tp = (1, [2, 6, 10], 3, 'name')
                    # [ ] is a List - another complex data type

>>> b_tp + c_tp                    # + operator, no -
('1', '2', '3', '4', '5', 1, [2, 6, 10], 3, 'name')

>>> b_tp*2                         # * operator, no /
```

?

Tuples, Lists in a tuple

```
>>> a1_tp = ((1, 2), 3, (4, 5, 6))     # Tuples () in a tuple ()
>>> a1_tp[2]
```

?

```
>>> c_tp = (1, [2, 6], 3, 'name')     # A List [] in a tuple ()
>>> c_tp[1]
```

?

List

A **compound** data type:

[0]

[1.2, 3.4]

[7, "Hi", "there", 99]

[]

List - Use [ ] to call items

```
>>> names = ["James", "Chen", "Kim", "Sergey"]
>>> names
>>> names[0]
'James'
>>> names[1]
'Chen'
>>> names[5]
Traceback (most recent call last):
IndexError: list index out of range
>>> names[-1]
'Sergey'
>>> names[-2]
'Kim'
>>> names[-4]
```

?

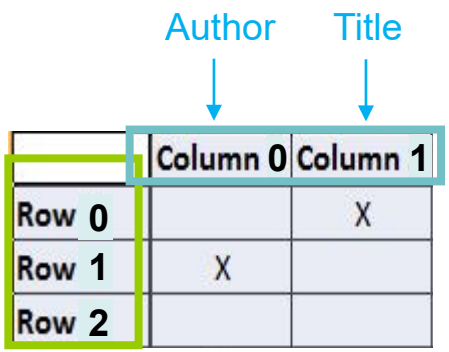
- ✓ Out of range values raise an exception
- ✓ Negative values go backwards from the last element.

Lists/ Tuples in a List
(*Nested* List/ Tuple)

```
a_lst = [['Tatiana', 'book I'], ['James', 'book II'], ['Sergey', 'book III']]
# It is a kind of 2-D array
# If we want to print each detail of a_lst[0] and a_lst[1]

print('Author:',a_lst[0][0])
print('Book title:',a_lst[0][1],'\n')

print('Author:',a_lst[1][0])
print('Book title:',a_lst[1][1],'\n')
```



Length of a *List*

```
>>> names = ["James", "Chen", "Kim", "Sergey"]
    # Use len() to get the length of a list

>>> len(names)          # length
4
>>> names[1]
```

?

List- Other useful *methods*

```
>>> ids = ["p0", "p1", "p2", "p3"]
>>> ids.append("p4")
    # to append the whole object, e.g. a string, tuple, list, to the list
>>> ids
['p0', 'p1', 'p2', 'p3', 'p4']

>>> id1=["abc", "def"]
>>> ids.extend(id1)
 # to extend the list by adding all the elements of an object to the end.
>>> ids
['p0', 'p1', 'p2', 'p3', 'p4', 'abc', 'def']

>>> ids.append(id1)                    # .extend vs .append
['p0', 'p1', 'p2', 'p3', 'p4',['abc', 'def'] ]
>>> ids[5]
['abc', 'def']
```

Exercise

```
bk1='Programming I' # book title
bk2='Programming II'
bk3='Programming III'
# bk1='Programming I'; bk2='Programming II'; bk3='Programming III'
a_lst=[ ]                    # We've created an empty bookshelf
a_lst.append(bk1)            # Put a book, bk1 on the bookshelf
a_lst.append(bk2)            # cf.   a_lst.extend(bk2)
a_lst.append(bk3)
print(a_lst[1])              # Show the item No.1

>>>'Programming II'
>>>a_lst
['Programming I', 'Programming II', 'Programming III']
```

Exercise

```
bk1=['Tatiana', 'book I', 1969]  # author, title, year
bk2=['John', 'book II', 1988]
bk3=['Ange','book III', 2019]

a_lst=[ ]
a_lst.append(bk1)  # Put an entire data set of the book information to a_lst[ ]
a_lst.?(bk2)
a_lst.extend(bk3)  # 3 books in a bookshelf

print(a_lst[1])
print(a_lst)            # It is a nested list, i.e. lists in a list.
[ ['Tatiana', 'book I', 1969], ['John', 'book II', 1988], ['Ange', 'book III', 2019] ]
```

List- Other useful methods

```
>>>del ids[5]        # The keyword (commend) del deletes an element
>>>ids
['p0', 'p1', 'p2', 'p3', 'p4']
>>>ids[2] = 'p8'
>>>ids.sort()       # sort by default order, i.e. 1,2,3,a,b,c
                     # (from the smallest to the largest)
>>>ids               # .sort is a method
['p1', 'p2', 'p3', 'p4']

>>>ids.reverse()                    # sort by reverse order
['p4', 'p3', 'p2', 'p1']
>>>ids.reverse()
['p1', 'p2', 'p3', 'p4']

        .insert(where, what)
>>>ids.insert(1, "p1.5")       # insert an object into a specific position
['p1', 'p1.5', 'p2', 'p3', 'p4']
>>>a_lst.insert(1,bk3)
>>>a_lst
```

* ***string*** and ***tuple*** also have their own **methods**.

Summary - *String, Tuple, List*

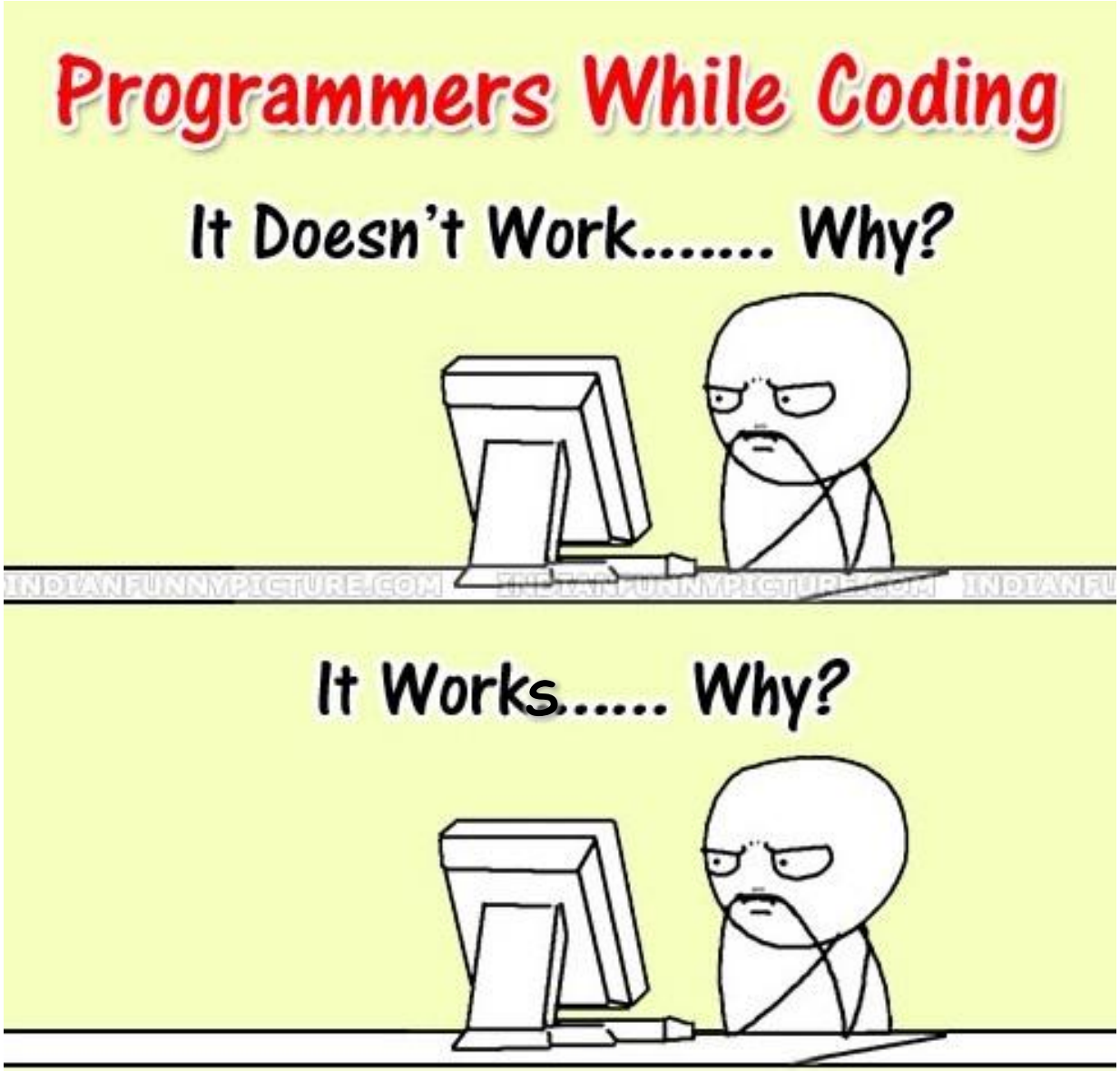
- **String** : character (text) string, ***immutable***
  - ✓ a_str = 'abcde'
  - ✓ a_str = '12345'
  - ✓ ~~a_str[1]=8~~ # illegal
- **Tuple** : composite data type, ***immutable***
  - ✓ a_tp = (1,2,3,4,5)
  - ✓ a_tp = 1,2,3,4,5
  - ✓ a_tp = tuple(a_str) # **type casting** a string to a tuple
  - ✓ ~~a_tp[1]=4~~ # illegal
  - ✓ a_tp= 1,2,'a','b','3pf','@email.com'
- **List** : composite data type, ***mutable***
  - ✓ a_lst = [1,2,'a','b','3pf','@email.com']
  - ✓ a_lst = list(a_tp) # **type casting** a tuple to a list
  - ✓ a_lst[1]=4 # legal, Okay

- Remember:

  we can't add elements to a tuple as it is **immutable**. Therefore, neither **.append()** nor **.extend()** method is available **for tuples,** but for lists.

Summary - Use of *" "*, *( )*, *[ ]*

- Assignment
  - ✓ string: " "
  - ✓ tuple: ( ) or no bracket
  - ✓ list: [ ]
- Calling a data element from a string/ tuple/ list
  - ✓ *data_structure_name*[*element_number*]
- Functions including the method
  - ✓ *func_name*(*inputs*)
  - ✓ .*method_name*(*inputs*)


Programmers While Coding
It Doesn't Work..... Why?
It Works..... Why?