


















# Python programming : basic grammar

ECE30007 Intro to AI Project

# outline

- getting started
  - install, jupyter
- basics
  - variable, control
- function
- object oriented programming
  - class
- modules
  - numpy
  - matplotlib

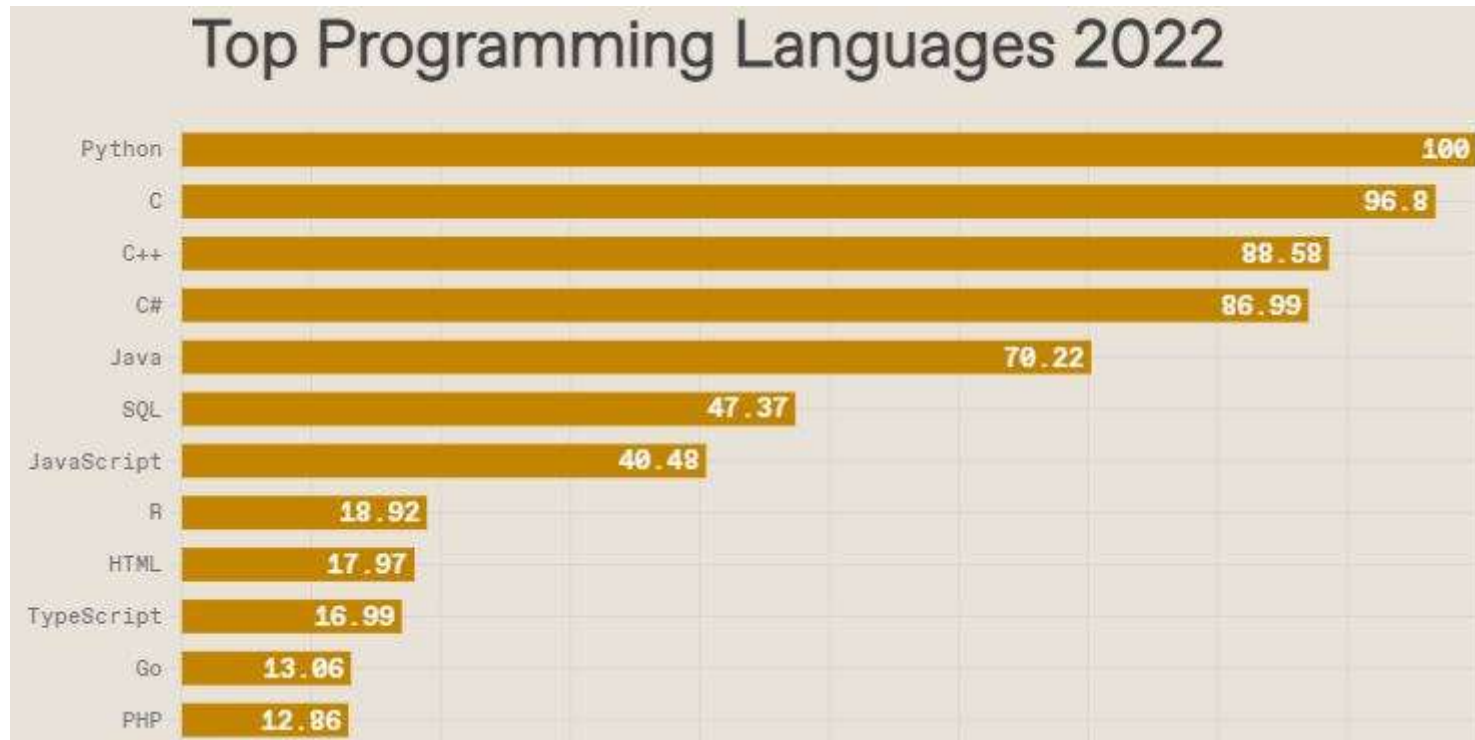
# IEEE Top 10 Programming languages of 2021

Rank	Language	Type	Score
1	Python	  	100.0
2	Java	  	95.4
3	C	  	94.7
4	C++	  	92.4
5	JavaScript		88.1
6	C#	   	82.4
7	R		81.7
8	Go	 	77.7
9	HTML		75.4
10	Swift	 	70.4

<https://spectrum.ieee.org/top-programming-languages-2021>



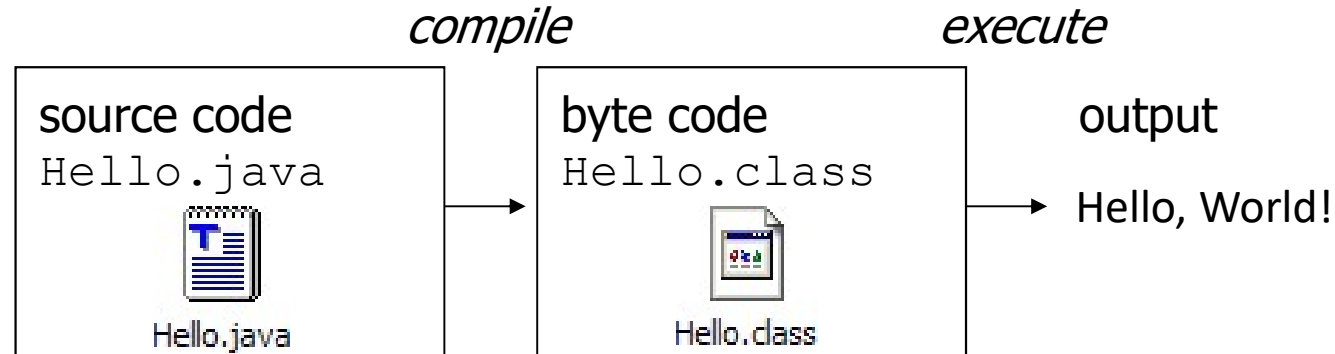
# IEEE Top 10 Programming languages of 2022



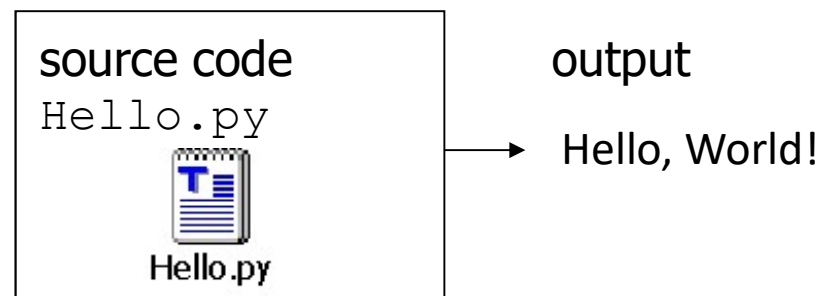
<https://www.i-programmer.info/news/98-languages/15695-ieee-spectrum-ranks-languages.html>

# compiling and interpreting

- Many languages require you to *compile* (translate) your program into a form that the machine understands.



- Python is instead directly *interpreted* into machine instructions.



# Python

- An interpreted, high-level, general-purpose programming language
  - Supports multiple programming paradigms, including procedural, object-oriented, and functional programming
  - Dynamically typed and garbage-collected
  - Emphasizes code readability  
(with its notable use of significant whitespace)
  - Written in C

# why Python?

- Clear syntax
  - *"Executable pseudo-code"*
- Easiness in data manipulation
  - Packages like NumPy, SciPy, Matplotlib, Pandas, ...
- Popularity — a large number of users and user groups
  - There is ample development and documentation
- Drawbacks
  - Not as fast as C or Java

# Python history

- Late 80s - Conceptualization and initial implementation
  - Led by Guido van Rossum (National Research Institute of Mathematics and Computer Science)
- 1991 - Python (version 0.9.0) was first released
  - classes, lists, strings, and exception handling
  - supports for functional programming (lambda, map, filter, and reduce)
- 2000 - Python 2.0 was released
  - Included list comprehensions and a fully-functional garbage collector
  - Started to support Unicode
- 2008 - Python 3.0 was released
  - Improved the internal mechanisms, grammar and expressions
  - Broke backward compatibility; *c.f.*, a tool called "2to3"
- Current versions
  - Python 3.10 (Mar 2022)



# Python Philosophy

- Python strives for a simpler, less-cluttered syntax and grammar
- Python embraces a "*there should be one — and preferably only one — obvious way to do it*" design philosophy
  - Beautiful is better than ugly
  - Explicit is better than implicit
  - Simple is better than complex
  - Complex is better than complicated
  - Readability counts
- Key features
  - Open source language
  - Extensive library support
  - Cross-platform

**"import this"**

# import this

In [1]: import this

The Zen of Python, by Tim Peters

("guiding principles" for writing computer programs)

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

# import this continued

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than \*right\* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

# install

<https://www.python.org/downloads/>



Looking for a specific release?

**python3.x is quite different from python2.x**

# hello world!

## in Terminal

```
Henrys-MacBook-Pro:~ henry$ python
Python 3.7.2 (v3.7.2:9a3ffc0492, Dec 24 2018, 02:44:43)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print('Hello World!')
Hello World!

In [2]: exit()
```

## if on Windows

- open a Python IDE like IDLE, and do the same thing above

## or IDE

- edit a python file (e.g., my\_code.py), and run it (e.g., python my\_code.py)

## or jupyter



# jupyter

install in Terminal

```
$ pip install jupyter
```

move to your working directory

```
$ cd ~/my_dir/
```

run jupyter

```
$ jupyter notebook
```

```
Henrys-MacBook-Pro:~ henry$ jupyter notebook --ip=127.0.0.1  
[I 01:07:20.256 NotebookApp] Loading IPython parallel extension  
(...)
```

To access the notebook, open this file in a browser:

file:///Users/henry/Library/Jupyter/runtime/nbserver-63283-open.html

Or copy and paste one of these URLs:

<http://127.0.0.1:8888/?token=47691093547ac0acf85af893e6417c6b6726d576>

open a web browser like Chrome and copy and paste the url

<http://127.0.0.1:8888/?token=47691093547ac0acf85af893e6417c6b6726d576>



# jupyter



Quit

Logout

Files

Running

IPython Clusters

Select items to perform actions on them.

Upload

New

Notebook:

Python 3

Other:

Text File

Folder

Terminal



jupyter Untitled1 Last Checkpoint: a few seconds ago (unsaved changes)

File

Edit

View

Insert

Cell

Kernel

Widgets

Help



Code



In [1]: `print('hello world!')`

hello world!

In [ ]:

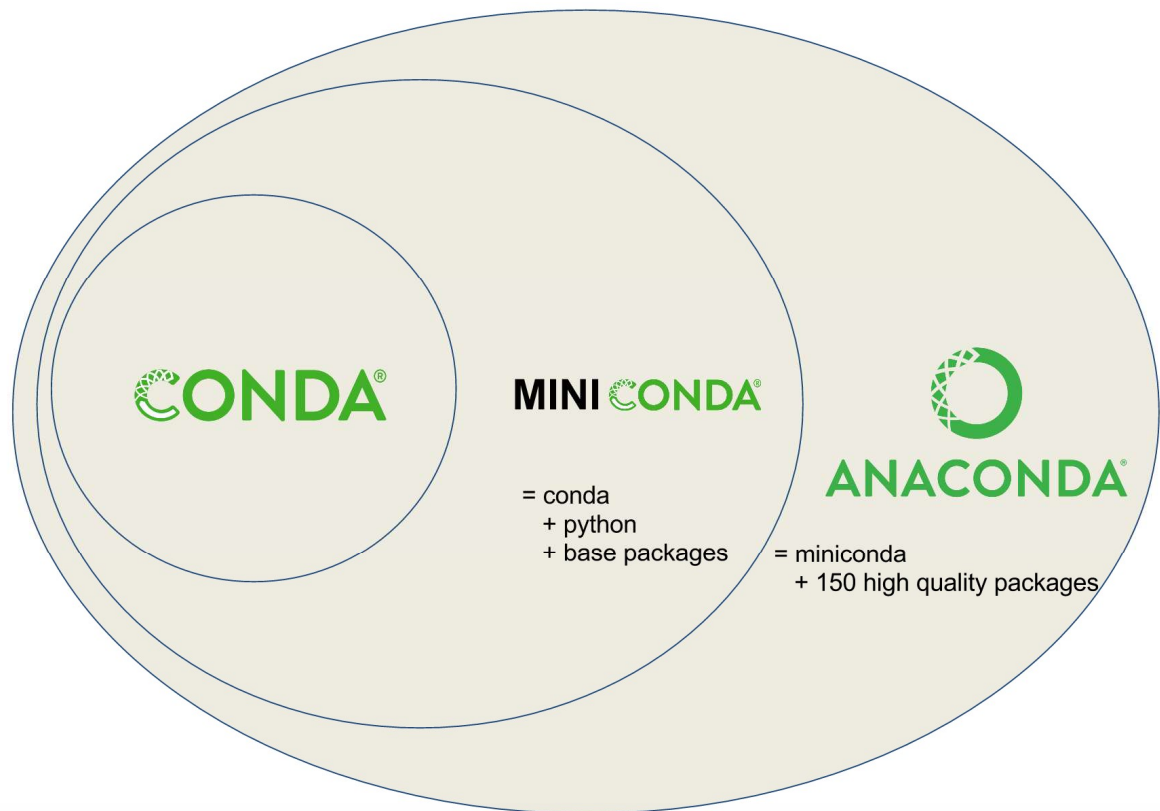


ECE30007 Intro to AI Project

# Conda <https://docs.conda.io/>

An open source package and environment management system that runs on Windows, Mac OS and Linux.


- Conda provides prebuilt packages or binaries.
- Conda is cross platform.
- Packages can be easily installed (through pip) in conda environment.





# Google colaboratory

<https://colab.research.google.com>

 Welcome To Colaboratory

File Edit View Insert Runtime Tools Help







Share 


Table of contents 


 Getting started


 Data science



 Machine learning


 More Resources

 Machine Learning Examples

 Section

+ Code + Text  Copy to Drive

Connect  Editing 



## What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch [Introduction to Colab](#) to learn more, or just get started below!

▼ **Getting started**

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ] seconds_in_a_day = 24 * 60 * 60
    seconds_in_a_day

86400
```

# indentation

- **indentation** instead of braces for the scope of expressions
- All lines must be indented the same amount to be part of the scope
  - or indented more if part of an inner scope
- use proper indentation
  - since the indenting is part of program.

```
In [2]: for x in range(1,10):  
        y = 2*x*x  
        y = y + 3*x  
        y = y + 5
```

```
In [3]: print(y)
```

194

the same number of spaces!

# comment

- compiler will ignore any thing after '#'

```
In [1]: print('comment') # this is comment  
        # this line is comment
```

comment

# variables

- no declaration
- the variable is created when you assign it a value
- everything is an object
  - variable, class, function, etc

# integer

- integer

```
In [2]: x = 3
```

```
In [3]: x
```

```
Out[3]: 3
```

- int(x) and float(x)

```
In [10]: int(3.5)
```

```
Out[10]: 3
```

```
In [11]: float(3)
```

```
Out[11]: 3.0
```

# integer

- the type of variable can change with the value

```
In [13]: x = 3.5  
         y = 2
```

type(x) is float

```
In [14]: x = y + 2
```

```
In [15]: type(x)
```

```
Out[15]: int
```

# string

- '+' is overloaded as in C++

```
In [16]: x = 'hello'
         y = 'world'
         z = x + ' ' + y
```

```
In [17]: z
```

```
Out[17]: 'hello world'
```

- some methods

```
In [23]: len(z), str(3.5), z[2], z[6:8]
```

# from index 6 to index 7  
# index starts from 0.

```
Out[23]: (11, '3.5', 'l', 'wo')
```

- immutable (e.g., `x[3] = 'X'` # illegal)

# string

`len(string)` - number of characters in a string  
(including spaces)  
`str.lower(string)` - lowercase version of a string  
`str.upper(string)` - uppercase version of a string

```
In [8]: len('HGU')
```

```
Out[8]: 3
```

```
In [9]: str.lower('HGU')
```

```
Out[9]: 'hgu'
```

```
In [10]: str.upper('Handong')
```

```
Out[10]: 'HANDONG'
```



# list

- ordered collection of data
- data can be of different types
- same subset operations as strings

```
In [100]: x = [2, 1, 4, 6]
          y = [1, 3.5, 'HGU']
          y[0:2]          # from index 0 to index 1
```

```
Out[100]: [1, 3.5]
```

```
In [101]: y[1:]          # from index 1 to the end
```

```
Out[101]: [3.5, 'HGU']
```

# slice operator

```
1 # Slice Operator
2 a = [1,2,3,4,5]
3
4 print(a[0:2]) # Choose elements [0-2), upper-bound noninclusive
5
6 print(a[0:-1]) # Choose all but the last
7
8 print(a[::-1]) # Reverse the list
9
10 print(a[::2]) # Skip by 2
11
12 print(a[::-2]) # Skip by -2 from the back
13
```

```
[1, 2]
[1, 2, 3, 4]
[5, 4, 3, 2, 1]
[1, 3, 5]
[5, 3, 1]
```

a

[0]	[1]	[2]	[3]	[4]
1	2	3	4	5

a[0:2]

[0]	[1]	[2]	[3]	[4]
1	2	3	4	5

## list: reassign (mutable)

- reassigns the  $i$ -th element

```
In [74]: x = ['i', 'love', 'you']
```

```
In [75]: x
```

```
Out[75]: ['i', 'love', 'you']
```

```
In [76]: x[2] = 'hgu'
```

```
In [77]: x
```

```
Out[77]: ['i', 'love', 'hgu']
```

## list: reference and copy

- y points x (they refer to the same object)
- if x changes, so does y

```
In [67]: x = ['i', 'love', 'you']
```

```
In [69]: y = x
```

```
In [72]: x[2] = 'hgu'
```

```
In [73]: y
```

```
Out[73]: ['i', 'love', 'hgu']
```

or, we can copy

```
In [111]: x = [1, 2, 3]
```

```
In [112]: y = x.copy()
```

```
In [113]: x[1] = 20
```

```
In [114]: y
```

```
Out[114]: [1, 2, 3]
```

```
In [115]: x
```

```
Out[115]: [1, 20, 3]
```

# list: append

- append

```
In [84]: x = [1, 2, 3, 4, 5]
```

```
In [85]: x.append(6)
```

```
In [86]: x
```

```
Out[86]: [1, 2, 3, 4, 5, 6]
```

```
In [87]: x = x + [7]
```

```
In [88]: x
```

```
Out[88]: [1, 2, 3, 4, 5, 6, 7]
```

# list: extend

- extend

```
In [90]: x = [1, 2, 3, 4, 5]
```

```
In [92]: x.extend([6, 7])
```

```
In [93]: x
```

```
Out[93]: [1, 2, 3, 4, 5, 6, 7]
```

```
In [94]: x = x + [8, 9]
```

```
In [95]: x
```

```
Out[95]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

# list: append

- append one more element

```
In [96]: x = [1, 2, 3, 4, 5]
```

```
In [97]: x.append([6, 7])
```

```
In [98]: x
```

```
Out[98]: [1, 2, 3, 4, 5, [6, 7]]
```

```
In [99]: x[5]
```

```
Out[99]: [6, 7] # [6, 7] is one element
```

```
In [14]: a = [1, 2, 3, 4]
```

```
In [15]: 1 in a
```

```
Out[15]: True
```

# tuples

A tuple is a collection which is ordered and unchangeable

```
In [102]: x = (1, 2, 3)
```

```
In [103]: x
```

```
Out[103]: (1, 2, 3)
```

```
In [104]: x[1] = 10          # immutable: cannot change the element
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-104-b88844ced749> in <module>  
----> 1 x[1] = 10  
  
TypeError: 'tuple' object does not support item assignment
```

```
In [105]: y = (2,)          # type is tuple
```

```
In [106]: y
```

```
Out[106]: (2,)
```

```
In [107]: y = (2)          # type is int
```

```
In [108]: y
```

```
Out[108]: 2
```

```
1 y = (2,)  
2 print(type(y))  
3  
4 y = (2)  
5 print(type(y))
```

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

why do we need tuple (immutable)?  
because it is faster than list (mutable).



# dictionary

- a set of key-value pairs
- mutable.

```
In [133]: x = {1:'paul', 2:'peter', 3:'john'}
```

```
In [134]: x[2]          # x.get(2)
```

```
Out[134]: 'peter'
```

```
In [135]: x[3]='james'
```

```
In [136]: x
```

```
Out[136]: {1: 'paul', 2: 'peter', 3: 'james'}
```

```
In [137]: x[5] = 'andy'
```

```
In [138]: del(x[3])
```

```
In [139]: x          # x.items()
```

```
Out[139]: {1: 'paul', 2: 'peter', 5: 'andy'}
```

# output

- print text output on the console

```
In [150]: print(x)
```

```
{1: 'paul', 2: 'peter', 5: 'andy'}
```

```
In [151]: print (x, y)      # y = [1, 2, 3]
```

```
{1: 'paul', 2: 'peter', 5: 'andy'} [1, 2, 3]
```

```
In [153]: age = 25  
print ('I am ', age, ' years old')
```

```
I am  25  years old
```

# output

- Print without newline

```
1 print("Hello there!")
2 print("It is a great day.")
3
4
5 print("Hello there!", end = '')
6 print("It is a great day.")
```

Hello there!  
It is a great day.  
Hello there!It is a great day.

- With format specifier

```
1 print('I am %d years old.' % 24)
2 print('I am %s.' % 'Handong')
3 print('Your score is %f' % 2.3)
4 print('%10s' % 'abcde')
5 print('Today is %d %s.' % (9, 'March'))
```

I am 24 years old.  
I am Handong.  
Your score is 2.300000  
abcde  
Today is 9 March.

# input

- reads a string value from user input

```
In [*]: age = input('how old are you?')
```

how old are you?

# string

```
In [164]: age = input('how old are you?')
```

how old are you?19      # after typing '19' in the blue box above

```
In [165]: print(age)
```

19

```
In [167]: int(age)+6
```

Out[167]: 25

# for loop

- with list, range

`range(start, stop)`

`range(start, stop, step)`

```
In [2]: for x in [1,3,5,9]:  
        print(x)
```

```
1  
3  
5  
9
```

```
In [3]: for x in range(5):  
        print(x)
```

```
0  
1  
2  
3  
4
```

```
In [4]: for x in range(3,7):  
        print(x)
```

```
3  
4  
5  
6
```

# for loop: enumerate

- One can easily find the index (iteration number) inside a "for" loop
  - Wrap an iterable with 'enumerate'
  - it will yield the item along with its index

```
1 # Know the index faster
2 vowels=['a','e','i','o','u']
3 for i, letter in enumerate(vowels):
4     print (i, letter)
5
```

```
0 a
1 e
2 i
3 o
4 u
```

# while loop

- while with a condition

```
In [6]: x = 1
        while x < 5:
            print(x)
            x = x + 1
```

1  
2  
3  
4

break, continue

- the same as in C

# if, elif, else

- `elif` means “else if”

```
In [7]: x = 10
        if x <= 5 :
            y = x + 5
        elif x <= 10 :
            y = x + 10
        else :
            y = x
        print (y)
```

20

no switch statement in Python  
→ instead, we can implement it.



# logic

- Many logical expressions use *relational operators*:

Operator	Meaning	Example	Result
==	equals	1 + 1 == 2	True
!=	does not equal	3.2 != 2.5	True
<	less than	10 < 5	False
>	greater than	10 > 5	True
<=	less than or equal to	126 <= 100	False
>=	greater than or equal to	5.0 >= 5.0	True

- Logical expressions can be combined with *logical operators*:

Operator	Example	Result
and	9 != 6 and 2 < 3	True
or	2 == 3 or -1 < 5	True
not	not 7 > 0	False

## exercise 1

- Write a program to print out a right triangle of '\*', whose width is given as an input.
- ex) when input is 5, the output should be as follows.

```
* * * * *
```

```
 * * * *
```

```
  * * *
```

```
   * *
```

```
    *
```

## exercise 2

- implement a program to print out a month

```
Enter the start day (0~6)>> 3
Enter the number of days (1~31) >> 31
Sun Mon Tue Wed Thu Fri Sat
      1  2  3  4
  5  6  7  8  9 10 11
 12 13 14 15 16 17 18
 19 20 21 22 23 24 25
 26 27 28 29 30 31
```

```
Enter the start day (0~6)>> 6
Enter the number of days (1~31) >> 29
Sun Mon Tue Wed Thu Fri Sat
                        1
      2  3  4  5  6  7  8
     9 10 11 12 13 14 15
    16 17 18 19 20 21 22
    23 24 25 26 27 28 29
```

# file I/O: read

```
In [14]: fp = open('input.txt', 'r')
         for line in fp:
             print(line)
         fp.close()
```

handong

global

university

```
In [16]: with open('input.txt', 'r') as fp:
         lines = fp.readlines()
         for line in lines:
             print (line)
```

```
In [15]: with open('input.txt', 'r') as fp:
         for line in fp:
             print (line)
```

handong

global

university

handong

global

university



# file I/O: write

```
In [17]: with open('output.txt', 'w') as fp:  
         fp.write('i love handong\n')  
         fp.write('i love pohang\n')
```

```
In [18]: cat output.txt
```

```
i love handong  
i love pohang
```



```
In [19]: fp = open('output.txt', 'w')  
         fp.write('i love hgu\n')  
         fp.write('i love korea\n')  
         fp.close()
```

```
In [20]: cat output.txt
```

```
i love hgu  
i love korea
```

## exercise 3

- read numbers from 'input.txt',  
and write the sum of each row to 'output.txt'  
and print out the sums

input.txt		output.txt		on screen
1.2 13 4.1		18.3		18.3
2 2 1 3		8		8
1 2 5 7 1 4		20		20
5.5 22		27.5		27.5

\* let's assume that we don't know how many lines we have in input.txt  
and how many numbers we have on each row

```
hint: In [29]: line = '1.2 13 4.1'
        line.split(' ')
        Out[29]: ['1.2', '13', '4.1']
```



## exercise 1 - solution

```
str_n = input('your input: ')
n = int(str_n)
for i in range(n):
    s = ''
    for j in range(i):
        s=s+' '
    for k in range(n-i):
        s=s+'*'
    print(s)
```



## exercise 2 - solution

```
sd = int(input('Enter the start day (0~6)>> '))
ndays = int(input('Enter the number of days (0~6)>> '))

str_days = 'Sun Mon Tue Wed Thu Fri Sat'
print(str_days)
for pos in range(sd):
    print('%3s ' % ' ', end='')

pos = pos + 1

for i in range(1,ndays+1):
    if pos == 6:
        print('%3s ' % i)
        pos = 0
    else:
        print('%3s ' % i,end='')
        pos = pos+1
```