



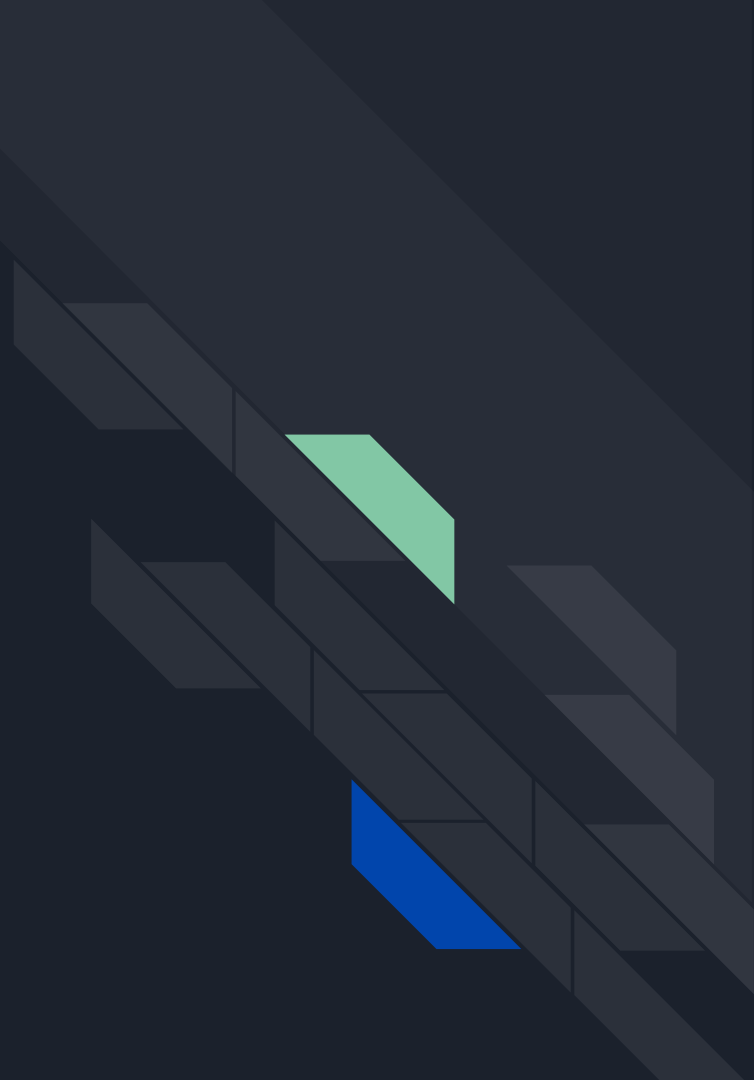
# Blueprints

Bhalaji N  
M.Tech., CSE (2016-2018)  
Amrita School of Engineering, Coimbatore

You are free to share, modify, copy and use this material and the associated python programs for educational purposes only.

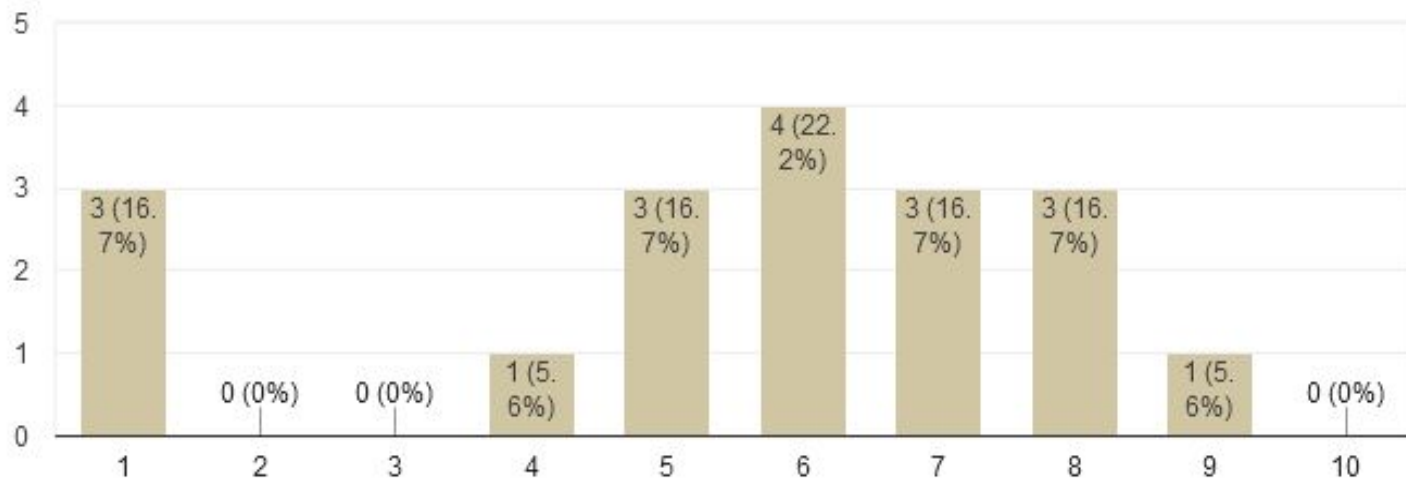
“Everybody should learn  
how to program a  
computer because it  
teaches you how to  
think”

- Steve Jobs



## How do you rate yourself in programming

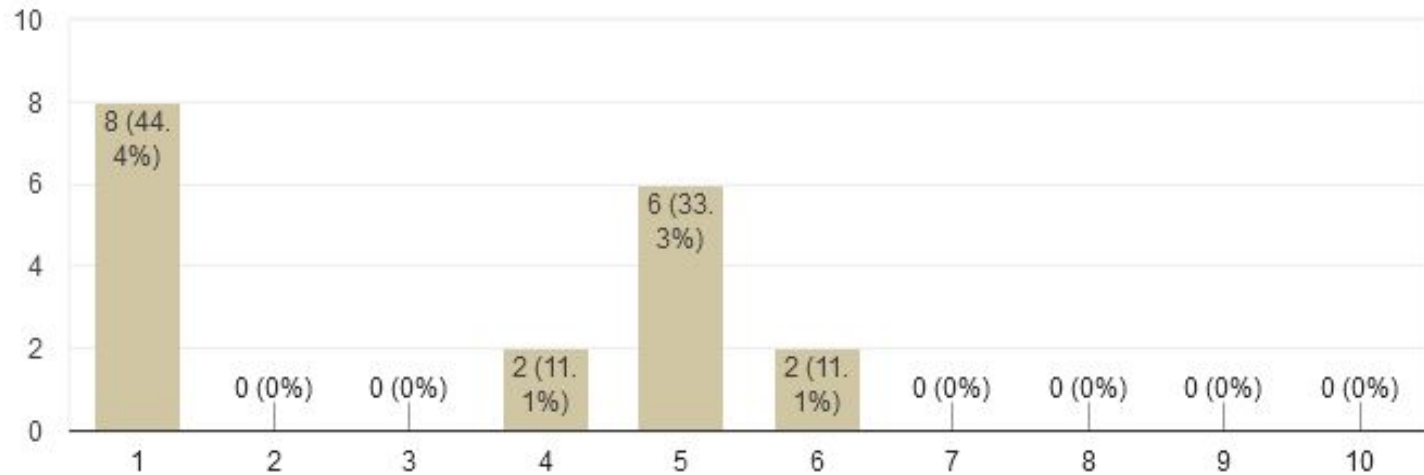
18 responses



\*Data collected from pre-workshop survey

## How do you rate yourself in python programming

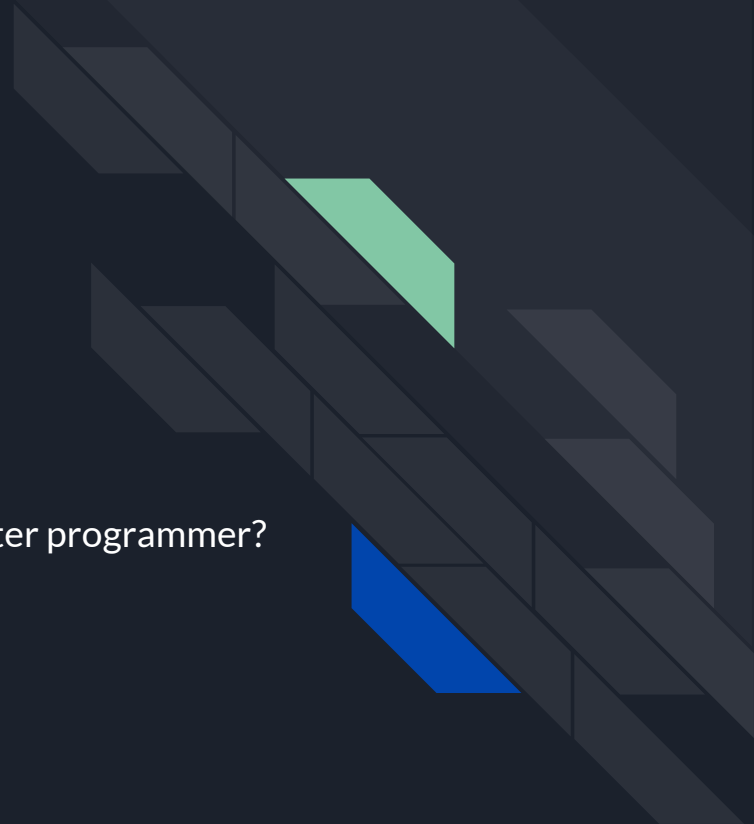
18 responses



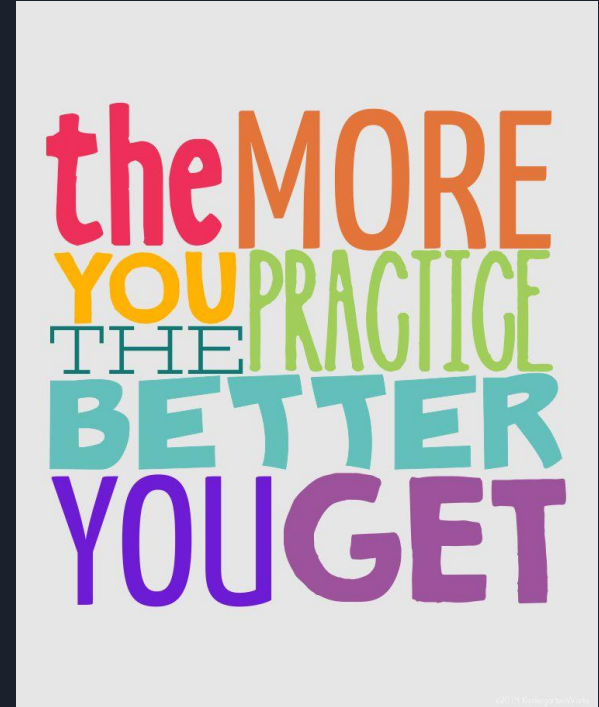
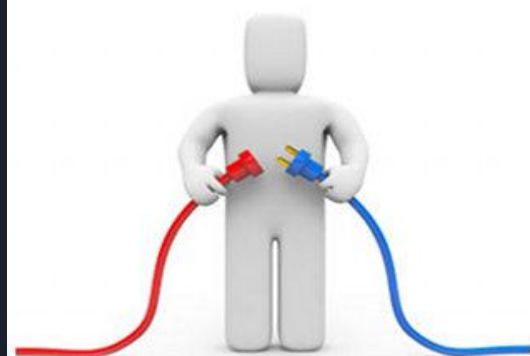
\*Data collected from pre-workshop survey

# Art of programming

What it takes to be a better computer programmer?



# Programming Recipe



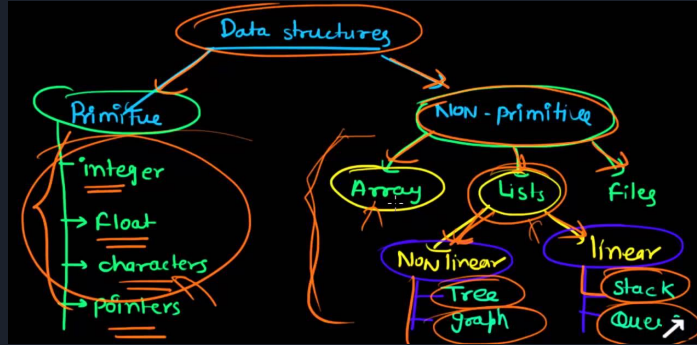


# Programming Recipe - Explained

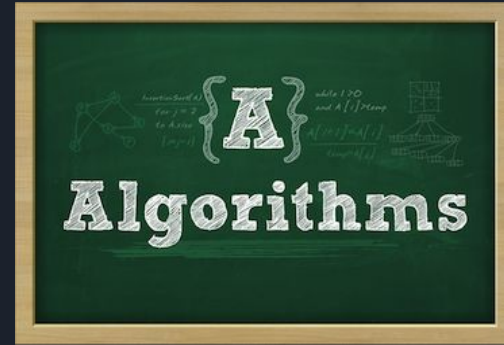
1. Visualization: Program and real word are not two different things. Visualize the problem as things and don't see them as programs
2. Relate: Program is just a tool. It is just a way to speak to computers. Relate the programs to day to day conversations
3. Programming is an art: It is not a skill, it is an art.
4. Cover your basics: Conceptual foundations needs to be strong.
5. Practice: Practice to become a fine artist

# Programming Recipe - The perfect blend

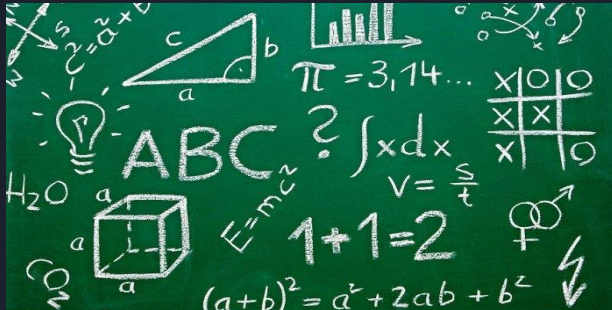
Data structures



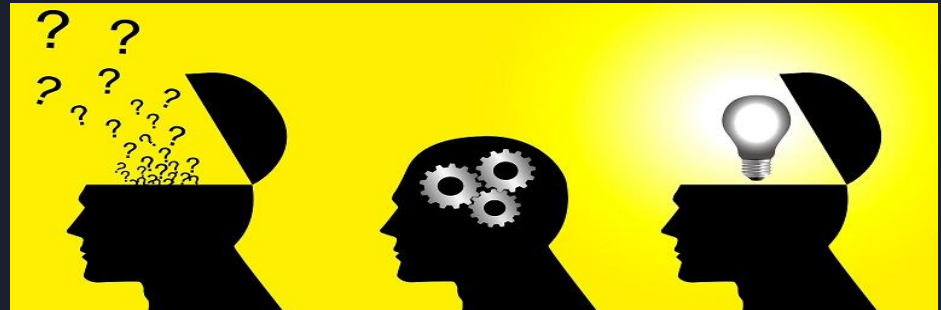
Algorithms



Mathematical foundations



Logical thinking and problem solving ability







# How to become a good programmer?

1. Look for where you are wrong?
2. Be open to learn
3. Don't stop at "the code works"
4. Read codes!!!
5. Learn techniques not tools
6. KISS
7. Analyse the problem
8. Work-around don't work
9. Code doesn't end with source file
10. Don't compare with other programmers

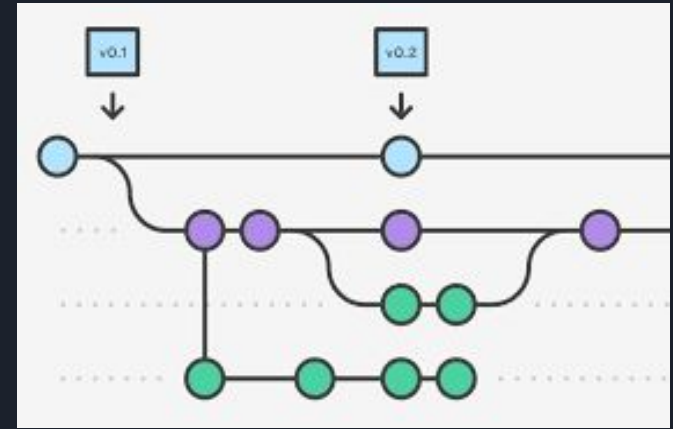


# Role of IDEs in effective programming

- Refactoring
  - Source Control
  - Reduction of learner's pain
  - Visual debugger
  - Auto complete
  - Centralization
- 
- Adaptation from "The Importance of the IDE in Software Development"

# Version Control Systems

- Manage source code over time
- Track every modifications to the code
- Benefits
  - Change history
  - Branching and merging
  - Traceability



Adaptations from

- “What is Version Control” by Atlassian Bitbucket

## Do It Yourself (DIY)

- “Learn Git with Bitbucket Cloud (Interactive tutorial)” by Atlassian Bitbucket



# Competitive Programming

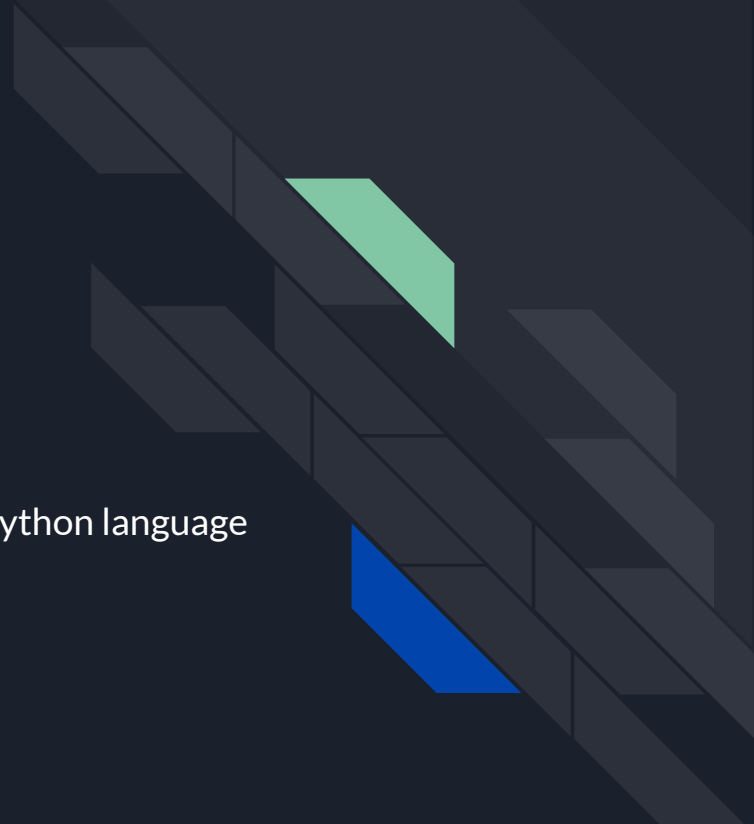
- Brings a blend of data structures, Algorithm Design and Analysis and Good problem-solving skills
- Competitive programming vs ML contests
- **Contest arenas**
  - Top coder, Code chef
  - Project Euler
  - Hacker Rank, Hacker Earth
- **ML Contests**
  - Kaggle
  - The AI Games

## Suggested Reads

- Want to ace technical interviews? Get started with Competitive Programming! - Aditya Rohilla
- The 10 most popular coding challenge websites for 2017 - Daniel Borowski
- What should every competitive programmer (Topcoder) know about Kaggle and ML contests? - Quora

# Python


















Introduction to Python language



Worldwide, Jul 2018 compared to a year ago:

| Rank | Change | Language    | Share   | Trend  |
|------|--------|-------------|---------|--------|
| 1    | ↑      | Python      | 23.59 % | +5.5 % |
| 2    | ↓      | Java        | 22.4 %  | -0.5 % |
| 3    | ↑↑     | Javascript  | 8.49 %  | +0.2 % |
| 4    | ↓      | PHP         | 7.93 %  | -1.5 % |
| 5    | ↓      | C#          | 7.84 %  | -0.5 % |
| 6    |        | C/C++       | 6.28 %  | -0.8 % |
| 7    | ↑      | R           | 4.18 %  | +0.0 % |
| 8    | ↓      | Objective-C | 3.4 %   | -1.0 % |
| 9    |        | Swift       | 2.65 %  | -0.9 % |
| 10   |        | Matlab      | 2.25 %  | -0.3 % |

PYPL Popularity of Programming

| Language Rank | Types   | Spectrum Ranking |
|---------------|---|------------------|
| 1. Python     |     | 100.0            |
| 2. C          |    | 99.7             |
| 3. Java       |    | 99.4             |
| 4. C++        |    | 97.2             |
| 5. C#         |    | 88.6             |
| 6. R          |    | 88.1             |
| 7. JavaScript |     | 85.5             |

| Jul 2018 | Jul 2017 | Change | Programming Language | Ratings | Change |
|----------|----------|--------|----------------------|---------|--------|
| 1        | 1        |        | Java                 | 16.139% | +2.37% |
| 2        | 2        |        | C                    | 14.662% | +7.34% |
| 3        | 3        |        | C++                  | 7.615%  | +2.04% |
| 4        | 4        |        | Python               | 6.361%  | +2.82% |
| 5        | 7        | ↑      | Visual Basic .NET    | 4.247%  | +1.20% |
| 6        | 5        | ↓      | C#                   | 3.795%  | +0.28% |
| 7        | 6        | ↓      | PHP                  | 2.832%  | -0.26% |
| 8        | 8        |        | JavaScript           | 2.831%  | +0.22% |
| 9        | -        | ↑↑     | SQL                  | 2.334%  | +2.33% |
| 10       | 18       | ↑↑     | Objective-C          | 1.453%  | -0.44% |

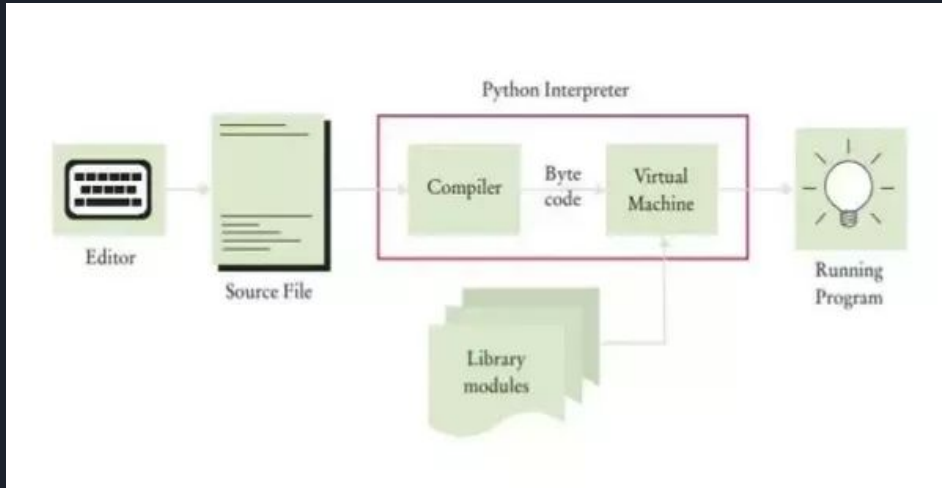
Tiobe-index



# Why is python so popular?

- Simple, ease of use, constantly improving
- High level language (still runs on any platform)
- Factors into consideration
  - Might be considered slow due to interpretation - 9/10 it doesn't matter
  - Presence of Global Interpreter Lock (GIL)

# Python interpreter



## Python is:

- Interpreted language
- Dynamically typed language
- Strongly typed language

Ref: Why is Python a dynamic language and also a strongly typed language? ([wiki.python.org](http://wiki.python.org))





# Application areas

- Integration and user tasks
- Scientific and Computational Applications
- Prototyping applications
- Web and internet development
- Software development
- Data mining and visualization
- Embedded systems
- Social science
- Machine learning and deep learning
- .... And a lots more....

# Python versions

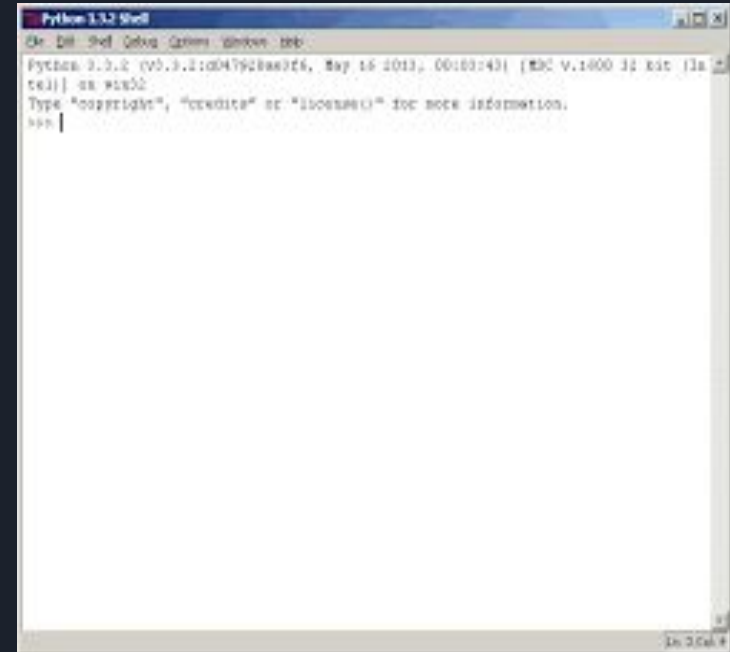
2.7.x vs 3.x

Detailed read:

- Should I use Python 2 or Python 3 for my development activity? ([wiki.python.org](http://wiki.python.org))

# Python IDLE and Python interpreter

- Basic ways of using Python
  - Python interpreter via command line
  - Python IDLE
- When using cmd
  - Write on any text editor
  - No separate steps for compilation
  - Just have to add python to path
  - Run the script using “python <script>.py”

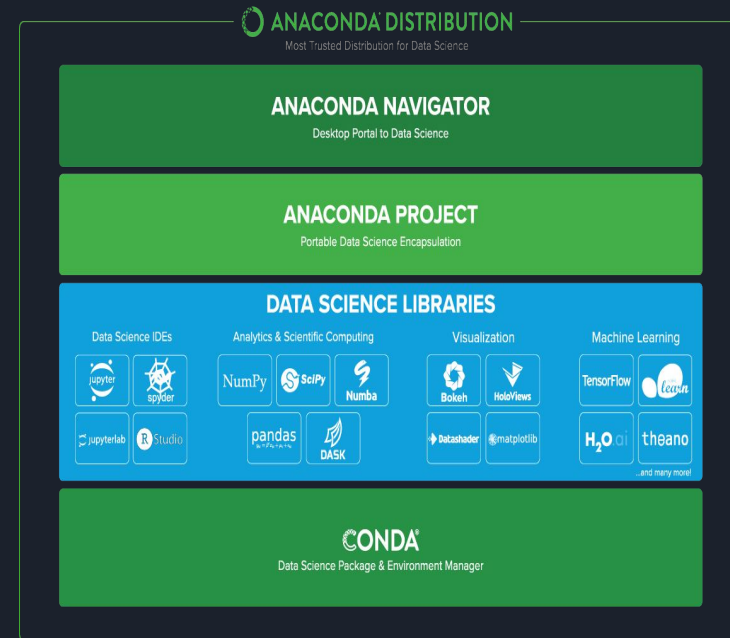
A screenshot of a Windows command prompt window titled "Python 3.3.2 Shell". The window shows the Python 3.3.2 shell prompt "Python 3.3.2 (v3.3.2:d047928ae2f6, May 16 2013, 00:03:43) [AMD64 v1.4000 32 bit (32-bit)] on win32". Below the prompt, it says "Type 'copyright', 'credits' or 'license()' for more information." and a cursor is visible on the line ">>>".

```
Python 3.3.2 Shell
Python 3.3.2 (v3.3.2:d047928ae2f6, May 16 2013, 00:03:43) [AMD64 v1.4000 32 bit (32-bit)] on win32
Type 'copyright', 'credits' or 'license()' for more information.
>>> |
```

# Anaconda



- Free and open source distribution for Python and R
- **Conda** - environment management and package management
- **Navigator** - Virtual environment manager (GUI)
- **Anaconda Cloud** - package management service





# Using third party modules


- Third party modules are available in plenty (PyPI)
- Installation is possible via conda and pip
- Pip (2008) succeed easy\_install (2004)
- Whl prebuilt versions for windows are available through other sources
- Conda and Pip are the most preferred

Notable reads:

- <https://stackoverflow.com/questions/3220404/why-use-pip-over-easy-install>

# Beginner's Programming

Python basics

An abstract graphic on the right side of the slide. It features a series of dark gray, three-dimensional rectangular blocks arranged in a stepped, diagonal pattern. A light green parallelogram is positioned on one of the upper blocks, and a blue parallelogram is on a lower block, both appearing to be part of the structure.

# Variables in Python

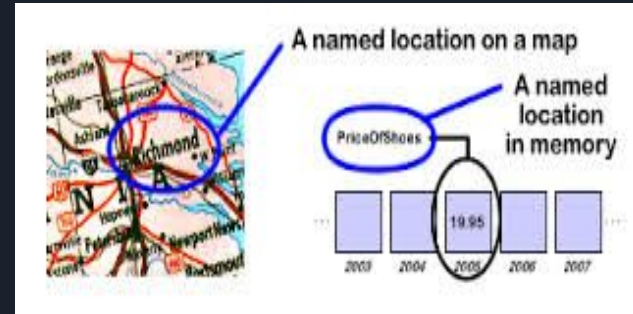
- Reserved memory locations to store values
- The interpreter allocates memory and decides what can be stored in the reserved memory
- The declaration happens automatically when you assign a value to a variable

## Multiple assignment

- `a = b = c = 1`
- `a, b, c = 1, 2, "john"`

## Suggested reads:

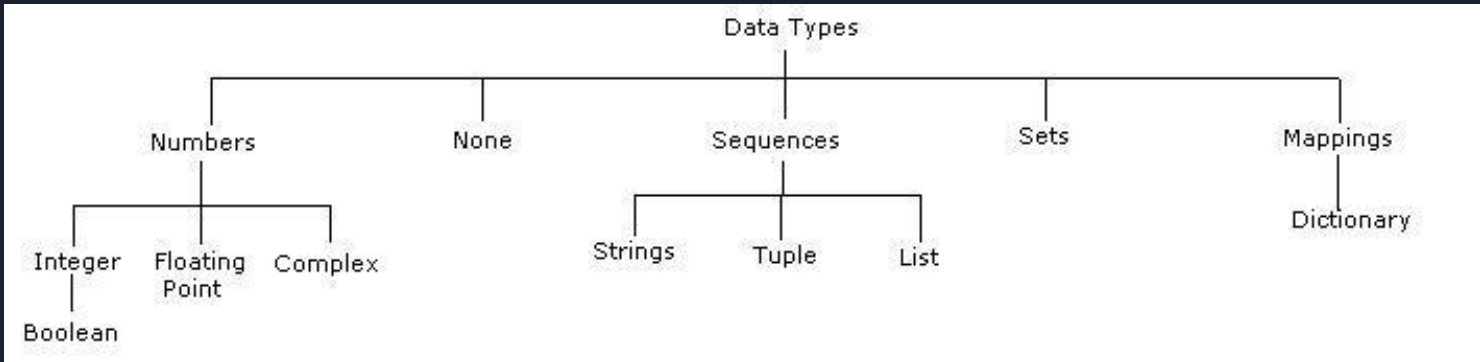
- Learn about `id()` function
- CPython Optimizations ([faster-cpython.readthedocs.io](https://faster-cpython.readthedocs.io))
- Python Variable Assignment Statements and Naming Rules & Conventions ([thehelloworldprogram.com](https://thehelloworldprogram.com))



# Data types

## Conversion between data types

- Convert between different data types by using different type conversion functions like `int()`, `float()`, `str()` etc





# Operators and precedence

| Operator                    | Description   |
|-----------------------------|---|
| **                          | Exponentiation (raise to the power)   |
| ~ + -                       | Ccomplement, unary plus and minus (method names for the last two are +@ and -@) |
| * / % //                    | Multiply, divide, modulo and floor division                                     |
| + -                         | Addition and subtraction  |
| >> <<                       | Right and left bitwise shift  |
| &                           | Bitwise 'AND'   |
| ^                           | Bitwise exclusive 'OR' and regular 'OR'   |
| <= < > >=                   | Comparison operators  |
| <> == !=                    | Equality operators  |
| = %= /= //= -= +=<br>*= **= | Assignment operators  |
| is is not                   | Identity operators  |
| in not in                   | Membership operators  |
| not or and                  | Logical operators   |



# Lists

- List of comma-separated values (items) between square brackets.
- Items in a list need not be of the same type.
- List indices start at 0, and lists can be sliced, concatenated and so on.

```
L = []
```

```
L = [expression, ..]
```

```
L = [expression for variable  
in sequence]
```

```
L = list()
```

```
L = list(sequence)
```



# List operations

- Accessing lists
  - `L[start:stop]`
- Slicing lists
  - `L[start:stop:slice]`
- Searching
  - `L.index(value)`
- Reversing
- Sorting
- Modifying lists
  - `L[i] = object`
  - `L[i:j] = sequence`
  - `append(item)`
  - `extend(sequence)`
  - `insert(index, item)`
  - `del L[i]`



# Tuples

- Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.
- You cannot update or change the values of tuple elements.
- **Tuple operations**
  - Length
  - Concatenation
  - Repetition
  - Membership
  - Iteration



# Sets

- Sets are lists with no duplicate entries
- Unordered collection of unique elements
- Sets are a powerful tool in Python since they have the ability to calculate differences and intersections between other sets.
- **Set operations**
  - Intersection
  - Difference
  - Symmetric\_difference
  - Union

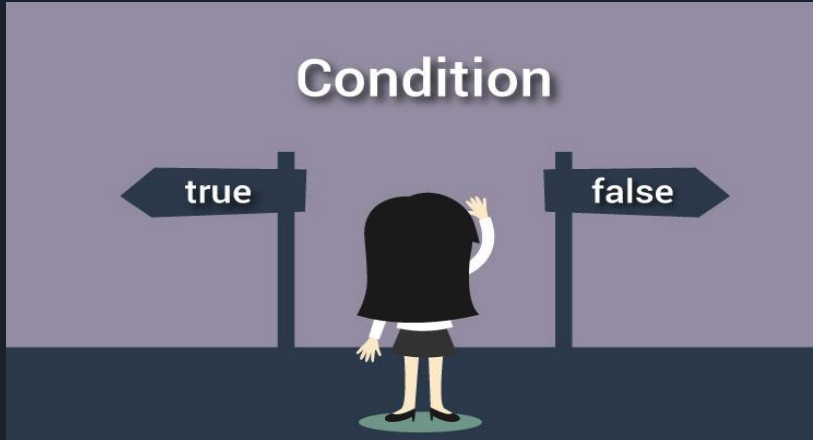
# Dictionaries

- Each key is separated from its value by a colon (:)
- The items are separated by commas
- The whole thing is enclosed in curly braces. {}
- Keys are unique within a dictionary while values may not be.
- Keys can be strings, tuples or numbers



**diy\_1\_dictionaries.py**

# Conditionals



```
if test expression:  
    body of if  
elif test expression:  
    body of elif  
else:  
    body of else
```

- Block scope is achieved via Indentation and not via parenthesis
- Elif ladder can be programmed for any level
- Nested if is possible for any level
- There is no switch case in Python

# DIY: Input statements

1. Write a program to convert temperatures to celsius from fahrenheit. ( $c/5 = (f - 32)/9$ )
  - a. Get input from user using input() function
  - b. Enter a prompt message when getting input from user
2. Expand the program to understandable format
3. Find the difference between eval(input()) and input()
4. Print the type of the input entered by the user
5. Validate the user input - input should only take float



**diy\_2\_input\_stmts.py**



# DIY: Conditionals

1. Write a program to check if the number is odd or even
2. Write a program to get the name of the day and print the number of the day in the week. (For example, if the user enters 'Sunday' print 'first day of the week')
  - a. Also handle inputs that are not days of the week and print 'invalid day'
  - b. Can you convert the el-if ladder to a dictionary??
3. Write a program that will generate a random playing card e.g. '9 Hearts', 'Queen Spades' when the return key is pressed.



[diy\\_3\\_conditionals\\_p1.py](#), [diy\\_3\\_conditionals\\_p2.py](#), [diy\\_3\\_conditionals\\_p3.py](#)



# Looping constructs

```
for LOOP_VARIABLE in SEQUENCE:  
    STATEMENTS
```

```
while BOOLEAN_EXPRESSION:  
    STATEMENTS
```

## Break statement

- The break statement is used to immediately leave the body of its loop.
- The next statement to be executed is the first one after the body

## Continue statement

- This is a control flow statement that causes the program to immediately skip the processing of the rest of the body of the loop, for the current iteration.
- But the loop still carries on running for its remaining iterations:

# DIY: Looping constructs

1. Write a Python program which iterates the integers from 1 to 50. For multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz". Hint: range() function
2. Define a procedure histogram() that takes a list of integers and prints a histogram to the screen. For example, histogram([4, 9, 7]) should print the following:

```
****
```

```
*****
```

```
*****
```



**diy\_4\_loops\_p1.py, diy\_4\_loops\_p2.py**



# List comprehension

Syntactic construct that enables lists to be created from other lists using a compact, mathematical syntax

```
[expr for item1 in seq1 for item2 in seq2 ... for itemx in seqx if cond]
```

```
output_sequence = []  
for item1 in seq1:  
    for item2 in seq2:  
        ...  
        for itemx in seqx:  
            if condition:  
                output_sequence.append(expr)
```



# List comprehension - Examples

```
>>> numbers = [1, 2, 3, 4]
```

```
>>> [x**2 for x in numbers]
```

```
[1, 4, 9, 16]
```

```
>>> [x**2 for x in numbers if x**2 > 8]
```

```
[9, 16]
```

```
>>> [(x, x**2, x**3) for x in numbers]
```

```
[(1, 1, 1), (2, 4, 8), (3, 9, 27), (4, 16, 64)]
```

```
>>> letters = ['a', 'b', 'c']
```

```
>>> [n * letter for n in numbers for letter in letters]
```

```
['a', 'b', 'c', 'aa', 'bb', 'cc', 'aaa', 'bbb', 'ccc', 'aaaa', 'bbbb', 'cccc']
```

```
>>>
```

# Strings


- Strings are created simply by enclosing characters in quotes.
- Python treats single quotes the same as double quotes.
- Creating strings is as simple as assigning a value to a variable
- String format operator - %s
- r'expression' and u'expression'
  - r - raw string (used in paths, regex, etc)
  - u - unicode string

## Suggested reads

- <https://learn.rmotr.com/python/understanding-unicode-in-python/strings-and-unicode/unicode-in-python>



**diy\_5\_strings.py**



# Regular expressions - A gateway for string matching

- Sequence of characters that define a search pattern
- Used by string matching algorithms
- Each character in regular expression is either a meta character or a literal character
- The module `re` provides full support for Perl-like regular expressions in Python
- Match and search functions



**`diy_6_regex_p1.py, diy_6_regex_p2.py`**



# Mutability of data types

- **Mutable data types**
  - Mutable objects can be changed after getting created
  - Lists are mutable data types
- **Immutable data types**
  - Immutable objects cannot be changed after creating
  - Tuples are immutable data types
- **Identify the mutability of other data types!!**



# Next level Programming





# Functions

```
def NAME( LIST OF PARAMETERS ) :  
    STATEMENTS
```

- Named sequence of instructions
- Use the new names without having to consider the details of the instructions to which they refer.

## Return statement

- The return statement causes a function to immediately stop executing statements in the function body
- Send back (or return) the value after the keyword return to the calling statement.

# Properties of functions

- Passing list as parameters
  - Call by reference and call by value - neither - it is call by assignment
- Functions as data
  - Storing functions as list
- Pure functions - Does not produce side effects, only result is in return statement
- Modifiers - A layer of packaging of the original function (Check decorators!!!)
- Recursive functions
- Polymorphism
- Duck typing



[diy\\_7\\_fns\\_purefns.py](#), [diy\\_7\\_fns\\_modifiers.py](#), [diy\\_7\\_fns\\_duck\\_type.py](#)



# Recursion

- Recursion is a way of programming or coding a problem, in which a function calls itself one or more times in its body.
- Usually, it is returning the return value of this function call.
- If a function definition fulfils the condition of recursion, we call this function a recursive function

# DIY: Functions

1. Write a Python program which accepts the radius of a circle from the user and compute the area.
2. Convert the weekday program as a function based program
3. Create a function that would take two arguments, weight and height and return BMI ( $BMI = \frac{xKG}{yM * yM}$ )
  - a. Convert the height as default parameter
  - b. Convert the second parameter as a variable argument list
    - i. Get age and height in the second parameter



**diy\_7\_fns\_p1.py, diy\_7\_fns\_p3.py**

# DIY: Functions

4. Create a function that would return the double of a parameter
  - a. Check polymorphism by passing different data types as a parameter
5. Write a recursive function to find factorial of a number (Try giving a very large number as input)
6. Create a list of functions and iterate each function with same parameters and observe the difference



**diy\_7\_fns\_p4.py, diy\_7\_fns\_p5.py**

# Modules and Packages

- A module allows you to logically organize your Python code.
- Simply, a module is a file consisting of Python code.
- A module can define functions, classes and variables.
- A module can also include runnable code.
- Packages are namespaces which contain multiple packages and modules themselves. They are simply directories
- Importing modules
  - `import module1[, module2[,... moduleN]`
  - `from modname import name1[, name2[, ... nameN]]`
  - `from modname import *`



**diy\_8\_modules.py**

# DIY: Modules and Packages

- Create a file called greetings.py
- Define a function called hello(name) and print “hello: name”
- Import the module greetings
- Call the function hello passing your name as parameter
- Move this to a folder called greetings (Note that if both greetings module and package cannot exist together)
- Create another function called bye(name)
- Add an empty file called \_init\_.py
- Try calling the bye function with your name



**diy\_8\_packages.py, hello.py, greetings (folder)**





# OS Module

- Miscellaneous operating system interfaces
- Working with directories and files
- Working with processes
  - Starting a new process
  - Forking a process
  - Running programs in background
- Invoking a daemon process

Suggested read:

- <http://effbot.org/librarybook/os.htm>

# SYS and CSV Modules

- **SYS Module**
  - System-specific parameters and functions
  - Command line arguments
  - Standard data streams
- **CSV Module**
  - Read and write tabular data
  - Reader and writer objects to read and write sequences



**diy\_9\_csv\_module.py**



# DIY: User input and file handling

1. The given folder contains different types of files
2. Display the available file names to the user (Use `listdir` in `os` module)
3. Ask user to enter name of a file (Use `input`)
4. Provide the extension of the file type (Use `os.path.splitext`)
5. Provide the size of the file (Use `os.path.getsize`)
6. Write the filename and size of the file to a csv file (Use `csv` module)
7. Write the filename and size of the file to a text file
8. Can you try modifying the user input to a command line argument?? (Use `sys.argv`)
9. Can you find other statistics of a file (`os.stat`)

# Classes

- Classes and objects
  - Classes are like modules
  - Objects are like imports
  - `__init__` method
  - Getters and setters
  - Access specifiers
- Inheritance
  - Is-A and has-A relationships

```
class ClassName:  
    <statement-1>  
    .  
    .  
    .  
    <statement-N>
```



[diy\\_10\\_classes\\_p1.py](#), [diy\\_10\\_classes\\_p2.py](#), [diy\\_10\\_classes\\_p3.py](#), [diy\\_10\\_classes\\_p4.py](#)



# DIY: Classes

- Define a class person with three attributes of different data types
- Create a object to the class
- Inherit the person class to employee class
- Add attribute salary to employee class
- Create object for employee class
- Try to retrieve the attributes of person class using employee class object (Check if upcasting and downcasting are possible?)

# Scopes and Namespaces

## Scopes

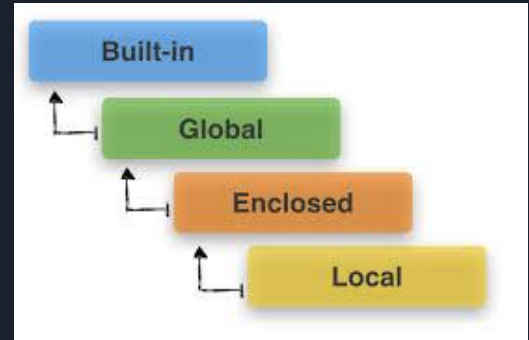
The scope of a definition is the part of the program over which the definition holds

## Namespaces

Roughly speaking, namespaces are just containers for mapping names to objects.

## Suggested Reads

- [http://www.python-course.eu/python3\\_namespaces.php](http://www.python-course.eu/python3_namespaces.php)
- <http://www.network-theory.co.uk/docs/pytut/PythonScopesandNameSpaces.html>
- [http://python-textbok.readthedocs.io/en/1.0/Variables\\_and\\_Scope.html](http://python-textbok.readthedocs.io/en/1.0/Variables_and_Scope.html)



LEGB Rule



**diy\_11\_scopes\_p1.py,  
diy\_11\_scopes\_p2.py,  
diy\_11\_scopes\_p3.py**

# Error and exception handling

- An exception is an error that happens during execution of a program.
- When that error occurs, Python generate an exception that can be handled, which avoids your program to crash.

```
raise [Exception [, args [,  
traceback]]]
```



**diy\_12\_exceptions.py**

```
try:  
    You do your operations here;  
    .....  
except ExceptionI:  
    If there is ExceptionI, then  
    execute this block.  
except ExceptionII:  
    If there is ExceptionII, then  
    execute this block.  
    .....  
else:  
    If there is no exception then  
    execute this block.
```

# DIY: Catch the exceptions

- Create a function to compute the sqrt of a number
- Use try...except block to catch TypeError
- Raise error to display a custom message when the number is 0



**diy\_12\_exceptions\_p1.py**



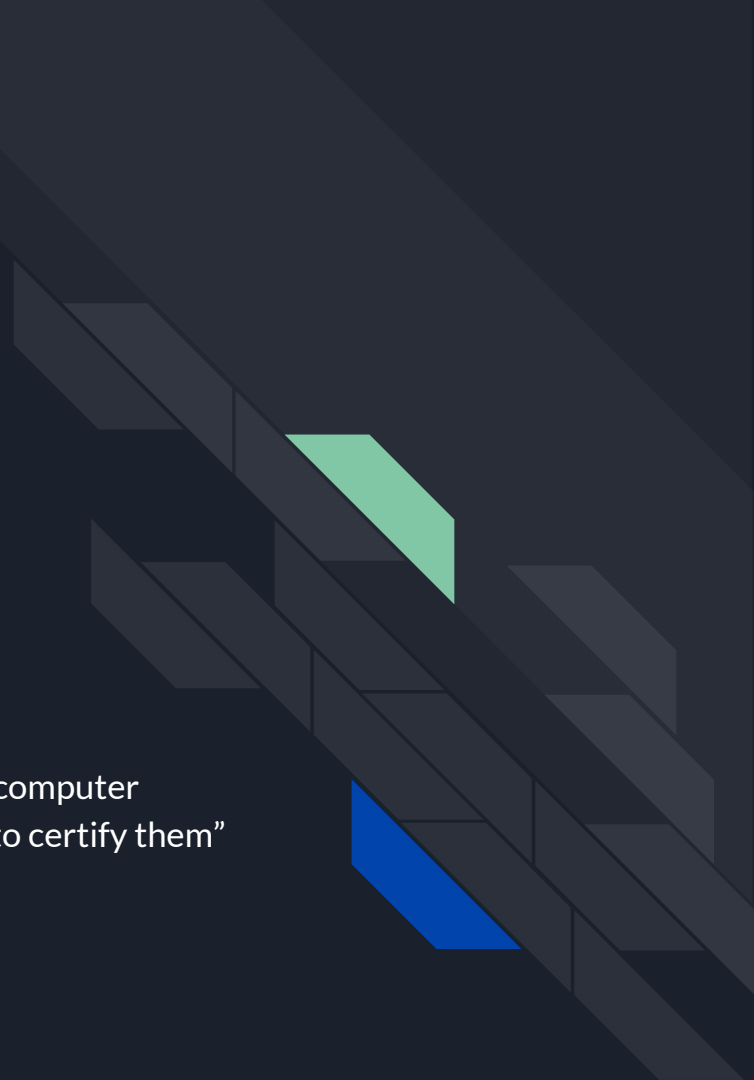


# Iterators and Generators

- Iterators
  - Returns a python iterator - one value at a time
  - Two methods are available: `__iter__()` and `__next__()`
- Generators
  - Create iterators using `yield`
  - Generators are used for lazy evaluations
  - Reusable generators are created using Object Oriented Generators
  - Generator expressions are used for high performance optimizations for list comprehensions

# Thinking like a programmer

“There are no standards for computer programmers and no group to certify them”





# Python debugger

- Debugging using command prompt
  - `python -m pdb <filename.py>`
- Debugging options
  - p - print
  - n - next
  - l - list
  - c - restart the execution
  - s - step
  - r - return

# DIY: Spot the errors and correct them

## Reference:

1. <http://tielvarolsson.com/blog/five-exercises-to-master-the-python-debugger/>
2. <https://docs.python.org/3/library/pdb.html>
3. <http://ericholscher.com/blog/2008/aug/30/using-pdb-python-debugger-django-debugging-series-/>
4. <https://pythonconquerstheuniverse.wordpress.com/2009/09/10/debugging-in-python/>



`diy_13_debugging_p1.py, diy_13_debugging_p2.py, diy_13_debugging_p3.py`



# Loggers

- Logging is important for system developing, debugging and running.
- When a program crashes, if there is no logging record, you have little chance to understand what happened.

Reference:

- <https://fangpenlin.com/posts/2012/08/26/good-logging-practice-in-python/>



**diy\_14\_loggers.py**



# Python practices

- Programming practices
  - PEP8 Style guide (<https://www.python.org/dev/peps/pep-0008/>)
  - BOBP Guide for Python (<https://gist.github.com/slوريا/7001839>)
- Zen of python - software principles that influence the design of Python Language
  - import this
- Structure of python projects
  - Filesystem structure of a Python project ([http://as.ynchronous.us/2007/12/filesystem-structure-of-python-project\\_21.html](http://as.ynchronous.us/2007/12/filesystem-structure-of-python-project_21.html))
  - Python Dev Workflow (<https://docs.pipenv.org/>)
- Scaffolding
  - PyScaffold (<https://pypi.org/project/PyScaffold/>)
  - Scaffold for Python (<https://github.com/Aaronontheweb/scaffold-py>)

# Python In and Out

Advanced Python Usage

An abstract graphic on the right side of the slide. It features several dark gray, 3D rectangular blocks arranged in a staggered, overlapping pattern. Two of these blocks are highlighted: one is a light green parallelogram and the other is a blue parallelogram, both appearing to be part of the structure.



# Python for Scientific Computing

## NumPy

- Fundamental package for scientific computing in Python
- Supports large, multi-dimensional arrays and matrices
- Holds several high level mathematical functions for matrix operations

## Scipy

- Module for optimization, linear algebra, integration, signal and image processing
- Works on numpy arrays
- Key algorithms and function cores for Python scientific computing capabilities





# Python for Scientific Computing

## Matplotlib

- Plotting library for Python
- API for embedding plots into applications using GUI Toolkits
- 2D plotting in publication quality fig with different formats and interactive environments

## Pandas

- Data manipulation and analysis library
- Provides data structures and operations for numeric tables and time series
- Highly optimized for performance (most of the critical code is written in Cython or C)



# Machine Learning in Python

## Scikit-learn

- Machine learning library for Python (data mining and data analysis)
- Classification, Regression and clustering algorithms
- Built on numpy, scipy and matplotlib

## Scikit-image

- Image processing library for Python
- Covers all basic algorithms for segmentation, geometric transforms, color space manipulations, feature detection, filtering, etc.
- Interoperates with NumPy and SciPy



# Image Processing in Python

## Pillow

- Python Imaging Library
- Support for multiple file formats with powerful image processing capabilities
- Ideal for batch processing and conversion of different file formats

## OpenCV-Python

- Python bindings to solve computer vision problems
- Infrastructure to build computer vision applications and to accelerate the use of machine perception in commercial products



# Interfaces with Python

## GUIs with TkInter

- Python binding to Tk GUI Toolkit
- Tkinter calls are translated to Tcl commands and are used by the Tcl interpreter available in the Python interpreter
- Positions of widgets within the containers are done using packers
- Application variables and Widgets are connected using the subclasses of Variable class
- Bind methods are used to watch for events and trigger callback functions
- Alternates for TkInter
  - wxPython, PyQt, Pygame, Pyglet, PyGTK



# Interfaces with Python

## Web interfaces with Flask

- Microframework for Python
- Supports extensions that can add application features.
- Flask does not have database abstraction layer, form validation by default
- Easy to setup and works well for smaller applications
- Has built in development server and debugger
- Alternatives for Flask
  - Django, Pyjs, Tornado

# Python Resources





# Books

- Learn Python - The Hard Way - EBook
  - <https://learnpythonthehardway.org/book/>
- A Byte of Python - EBook
  - <https://python.swaroopch.com/>
- Think Python: How to Think Like a Computer Scientist - EBook (PDF/HTML)
  - <http://greenteapress.com/thinkpython/html/index.html>
- Intro CS - Harvey Mudd College - Lecture Series (This covers more than programming)
  - <https://www.cs.hmc.edu/csforall/>



# Video Lectures

- Learn Python - Codecademy
  - <https://www.codecademy.com/learn/learn-python>
- Google for Education - Python
  - <https://developers.google.com/edu/python/>
- Intro to Python for Data Science - Video lectures
  - <https://www.datacamp.com/courses/intro-to-python-for-data-science>
- MIT 6.00SC Introduction to Computer Science and Programming (26 Lectures)
  - <https://www.youtube.com/playlist?list=PLB2BE3D6CA77BB8F7>





# Online Resources

- The Python Tutorial
  - <https://docs.python.org/3/tutorial/index.html>
- Python Tricks - Newsletter - short python code snippets
  - <https://realpython.com/python-tricks/>
- Learn X in Y minutes - Where X=Python
  - <https://learnxinyminutes.com/docs/python/>
- List of Python Frameworks, Libraries, Softwares and Resources
  - <https://github.com/vinta/awesome-python>



# Python Challenge

- <http://www.pythonchallenge.com/>
- Python based online programming riddle
- All levels have to be solved using Python Scripts
- Helps you explore the language in an entertaining way
- [Form groups and try to crack the challenge](#)

That's all folks :) :)

Have a wonderful career

