Getting Started with Machine Learning

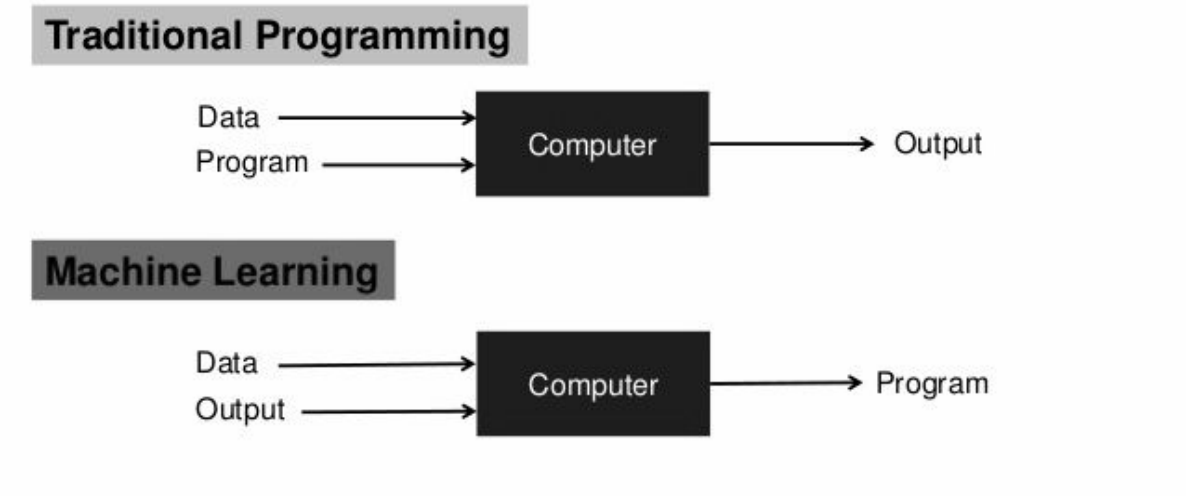
Who am I?

- MLE.
- Entrepreneur.
- Teacher.
- Competitive Programmer.
- Open Source Contributor.

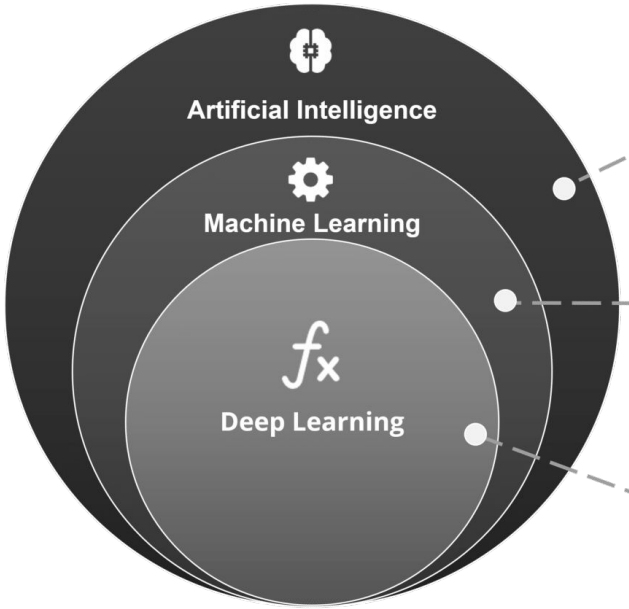
Connect with me on LinkedIn:
https://www.linkedin.com/in/nagarajubudigam/

What is Machine Learning?

How is it different from traditional programming?



AI vs. ML vs. DL

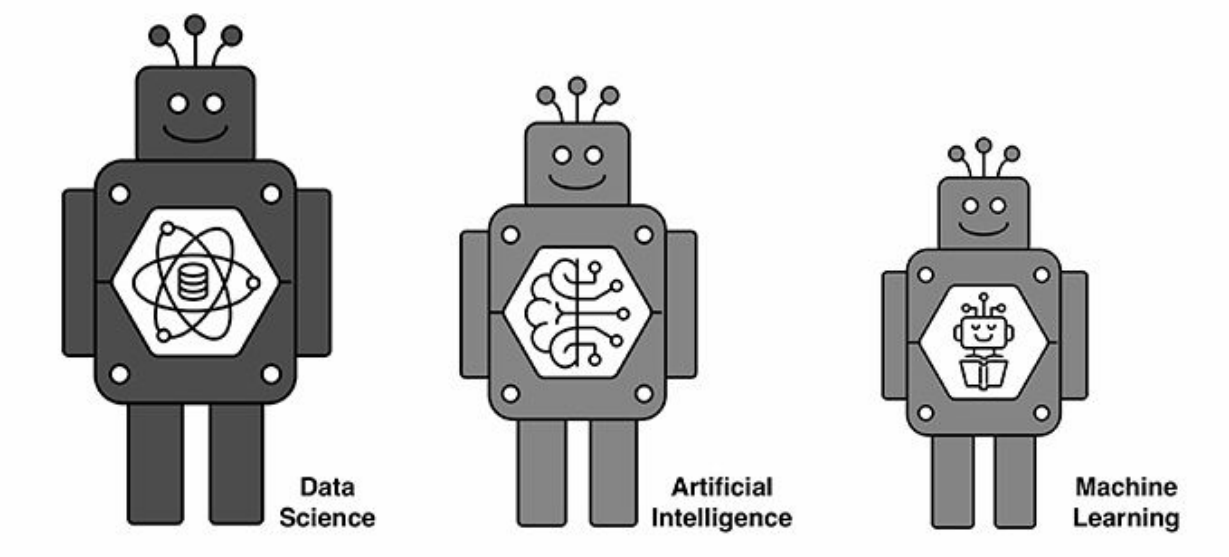


ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour

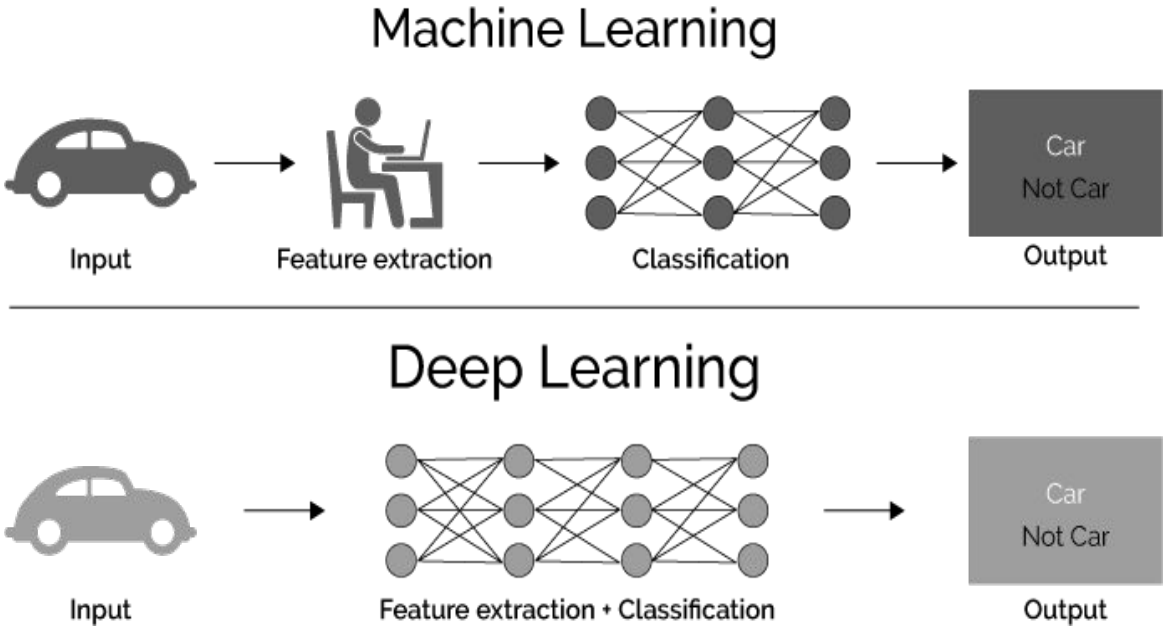
MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience

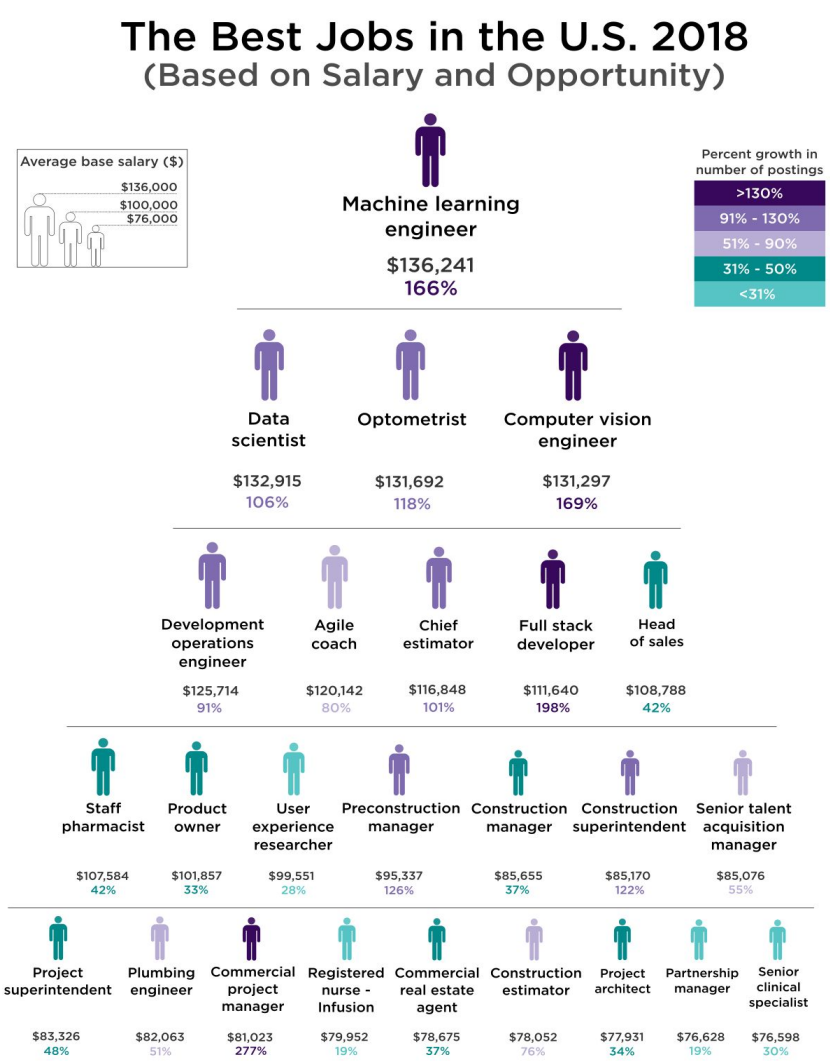
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Data Science vs. AI vs. ML



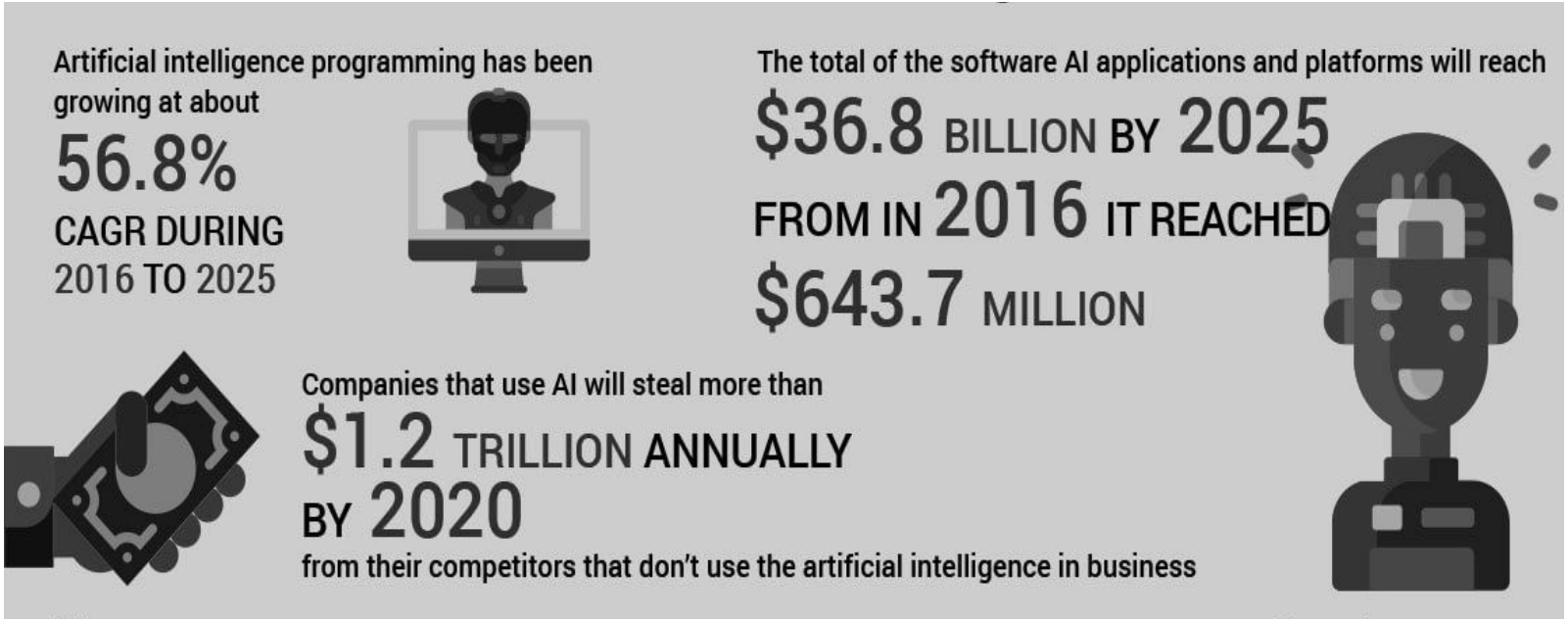
ML vs. DL


Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Deep Learning
Input
Feature extraction + Classification
Car
Not Car
Output



Why should you learn Machine Learning?

The Future of Artificial Intelligence



The Myths

- Data Science is **for genius mathematics brains** of Ph.D holders.
- Data Science is **all about learning tools only**.
- Data Scientist **jobs will be grabbed by AI soon**.
- Data Scientists **work on sophisticated tools** all the time.
- Data Scientists **need to work only on bulk data**.
- You **need to know coding then only you can become a Data Scientist**.
- **Data Science and Business intelligence are same**.

Course Timeline

07 Unsupervised Learning

- Clustering
- Feature Scaling
- Feature Selection
- PCA
- Feature Transformation

06 Supervised Learning

- Tree Based Models
- Probabilistic Models
- Linear Models
- Kernel Based Models
- Neural Nets

05 H2o ML Framework

- Setting up H2o
- Common Model Parameters
- Model building & Evaluation
- Hyper Parameter Tuning
- Model Export

04 Machine Learning Foundation

- Fundamentals of ML
- Model Selection
- Testing Models
- Evaluation Metrics
- Error Detection

03 Probability

- Terms Used in Probability
- Multiplication Rule of Probability
- Types of Events
- Bayes Theorem

02 Statistics

- Terms used in Statistics
- Types of Statistics
- Common Data Distributions

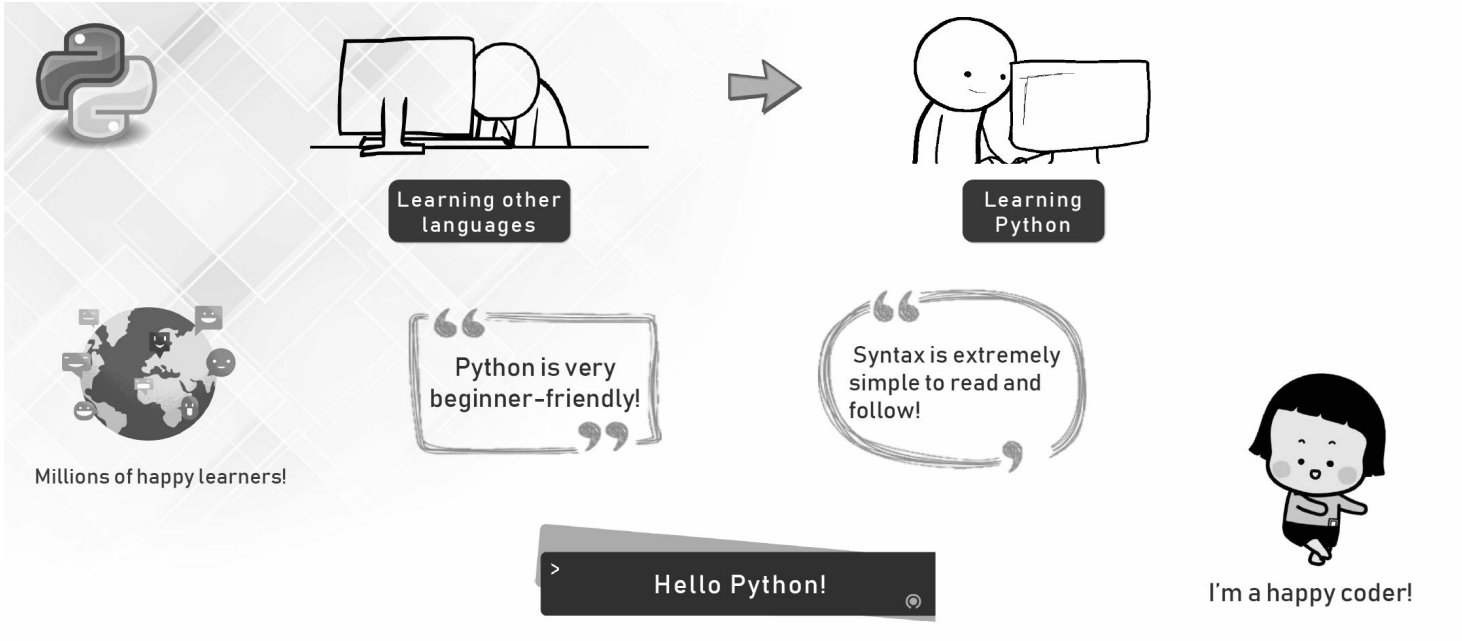
01 Python Programming

- Core Concepts of Python
- Object Oriented Programming with Python

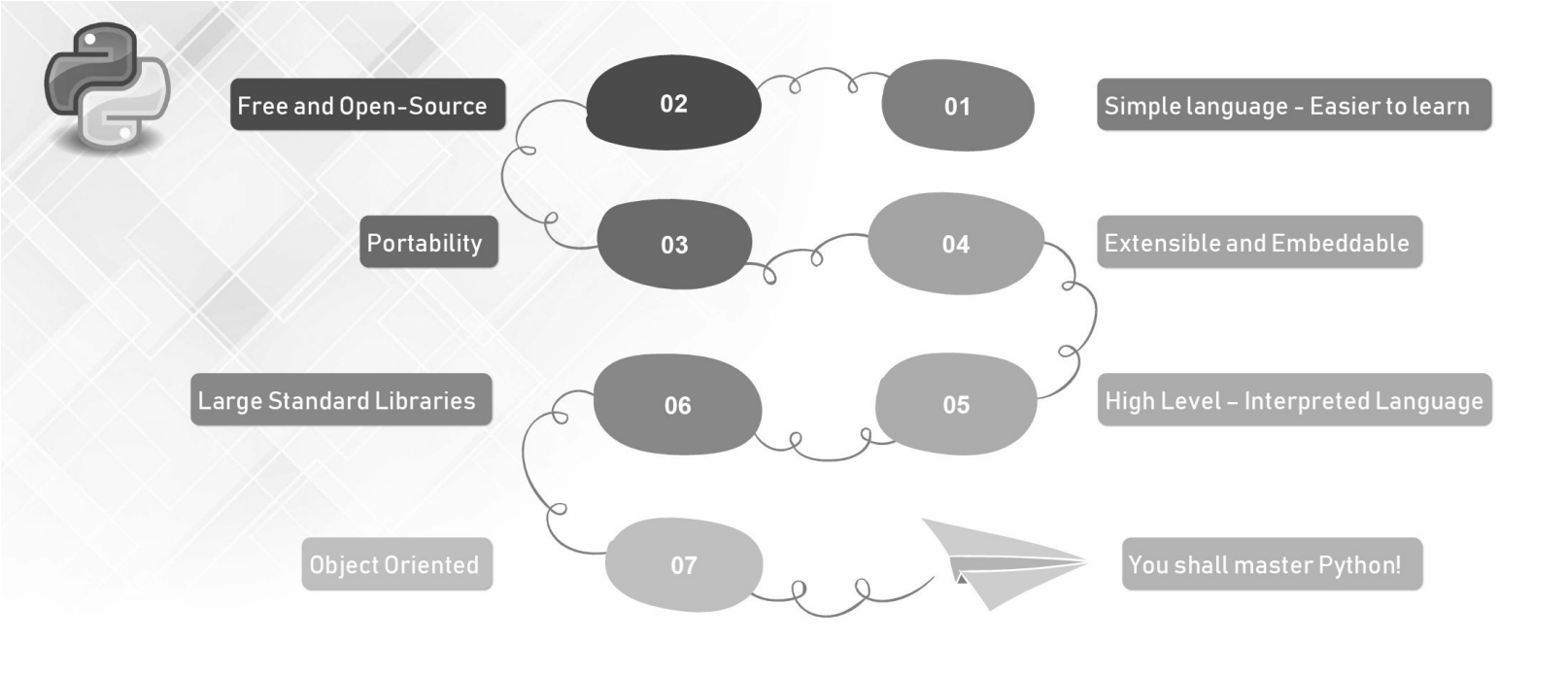
Let’s Get Started with Python Programming

Sometimes we all need a little motivation.

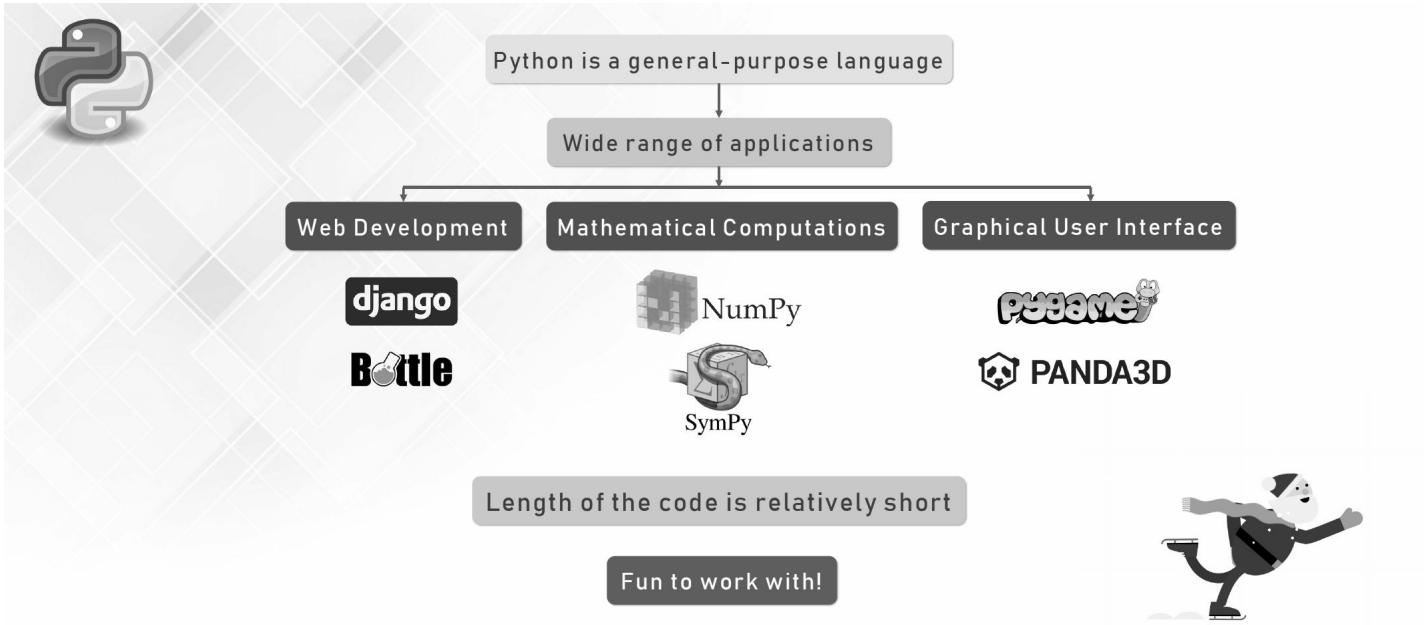
Why Python is so popular?



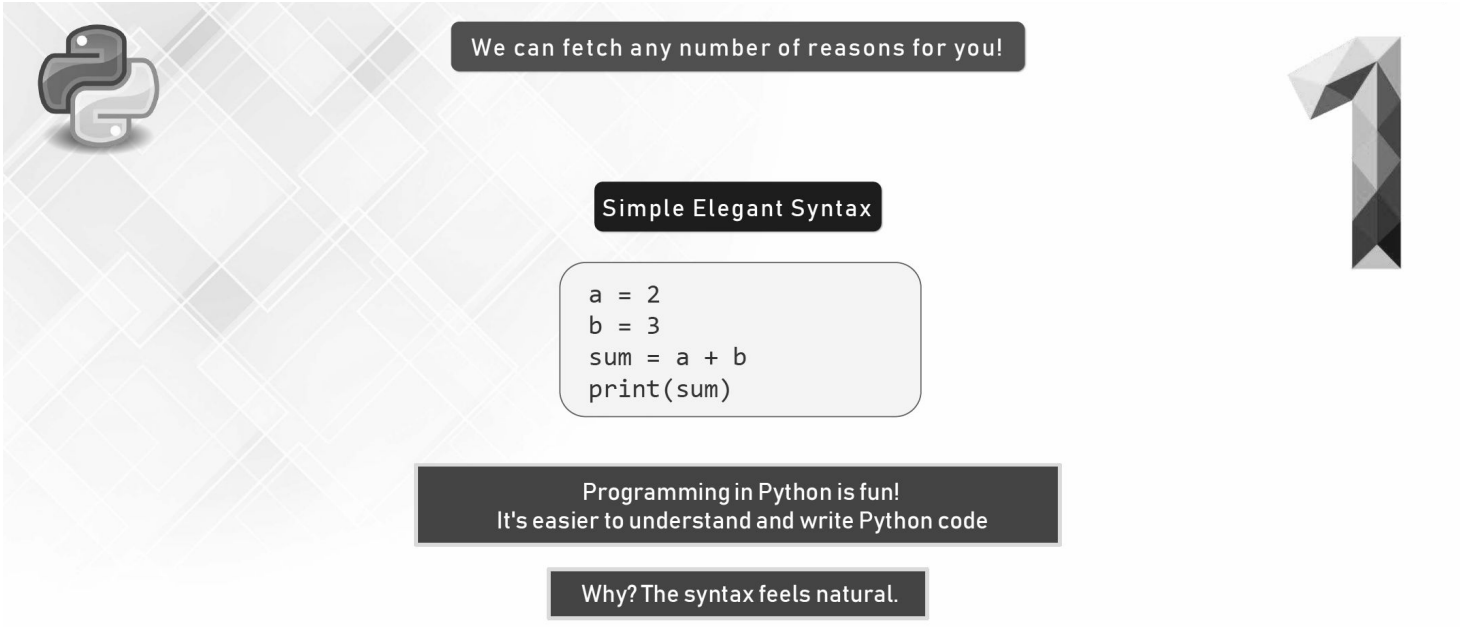
What are the features of Python?



Why should you learn Python?



What are the reasons to choose Python?



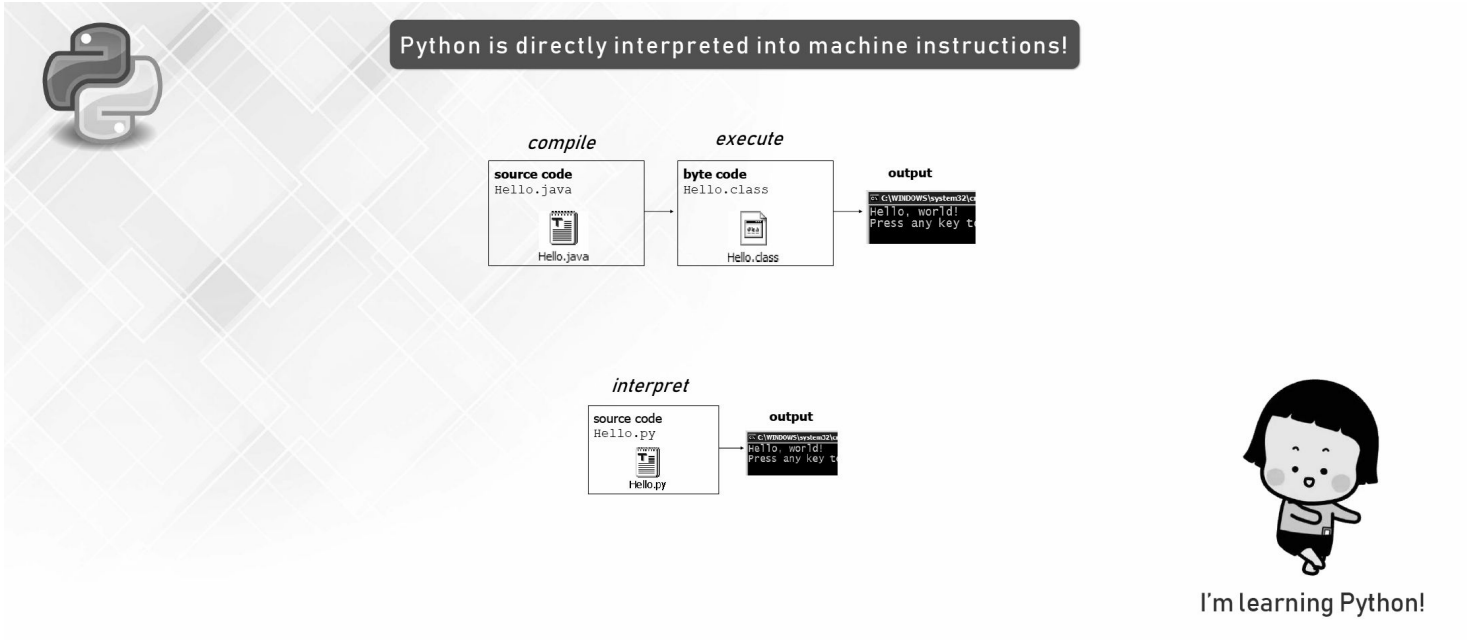
What are the reasons to choose Python?

What are the reasons to choose Python?

What are the reasons to choose Python?

Who is using Python?

What is the difference between compiling and interpreting?



How Python is different from other languages?

Dynamically Typed - Type of Data is inferred during run time

Development Environments

https://www.anaconda.com/download

Download Anaconda Distribution

Version 2018.12 | Release Date: December 21, 2018

Download For:

Anaconda 2018.12 For macOS Installer

Python 3.7 version *

Download

64-Bit Graphical Installer (652.7 MB)

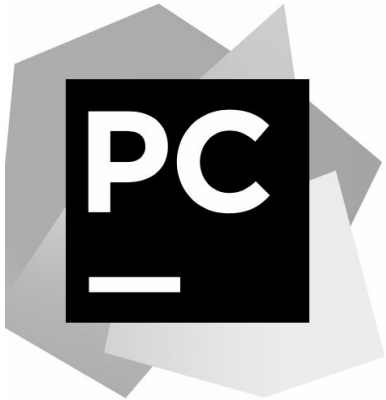
64-Bit Command-Line Installer (557 MB)

Python 2.7 version *

Download

64-Bit Graphical Installer (640.7 MB)

64-Bit Command-Line Installer (547 MB)



Download PyCharm

Windows | macOS | Linux

Version: 2018.3.3

Build: 183.5153.39

Released: January 10, 2019

System requirements

Installation Instructions

Previous versions

Professional

Full-featured IDE
for Python & Web
development

DOWNLOAD

Free trial

Community

Lightweight IDE
for Python & Scientific
development

DOWNLOAD

Free, open-source

https://www.jetbrains.com/pycharm/download

How to install Python packages? (Optional)

pip

- Python packages only.
- Compiles everything from source. EDIT: pip now installs binary wheels, if they are available.
- Blessed by the core Python community (i.e., Python 3.4+ includes code that automatically boostraps pip).

conda

- Python agnostic. The main focus of existing packages are for Python, and indeed conda itself is written in Python, but you can also have conda packages for C libraries, or R packages, or really anything.
- Installs binaries. There is a tool called `conda build` that builds packages from source, but `conda install` itself installs things from already built conda packages.
- External. Conda is the package manager of Anaconda, the Python distribution provided by Continuum Analytics, but it can be used outside of Anaconda too. You can use it with an existing Python installation by pip installing it (though this is not recommended unless you have a good reason to use an existing installation).

How to install Python packages?

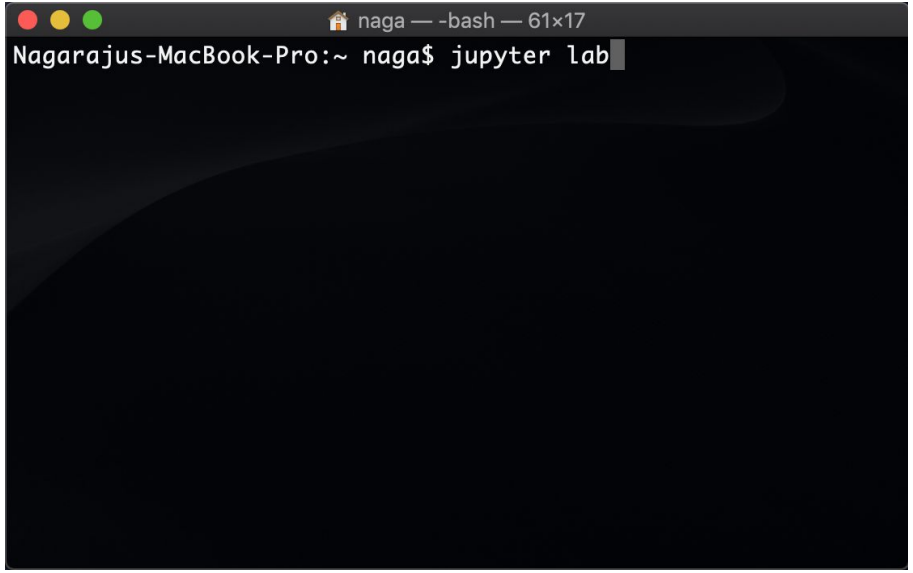
pip

- pip **install** *<package-name>*
- pip **uninstall** *<package-name>*
- pip **install** --upgrade *<package-name>*
- To install a specific version of a package
  - pandas install <package-name>==<version>

conda

- conda **install** *<package-name>*
- conda **uninstall** *<package-name>*
- conda **update** *<package-name>*
- To install a specific version of a package
  - conda install <package-name>==<version>
- Cheat Sheet: https://kapeli.com/cheat_sheets/Conda.docset/Contents/Resources/Documents/index

Let's start Jupyter Lab



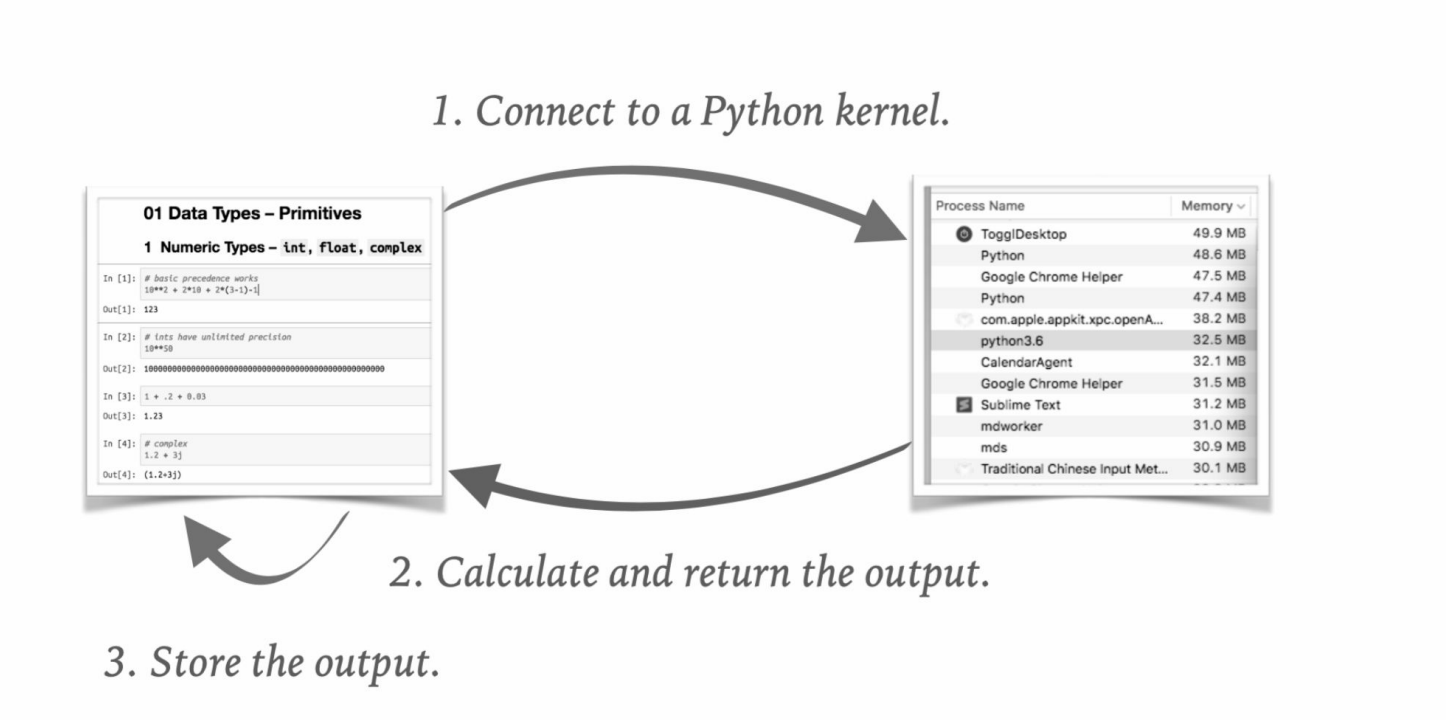
Common Jupyter Lab Shortcuts

`Esc`	Edit mode → command mode.
`Ctrl-Enter`	Run the cell.
`B`	Insert cell below.
`D, D`	Delete the current cell.
`M`	To Markdown cell.
`Cmd-/`	Comment the code.
`H`	Show keyboard shortcuts.
`P`	Open the command palette.

Common Markdown Syntax

`# Header 1`	Header 1
`## Header 2`	Header 2.
`> Block quote`	Block quote.
`* Item 1` `* Item 2`	Unordered list.
`1. Item 1` `2. Item 2`	Ordered list.
`*emphasis*`	*Emphasis.*
`**strong emp**`	**Strong emphasis.**

How Jupyter Lab works?

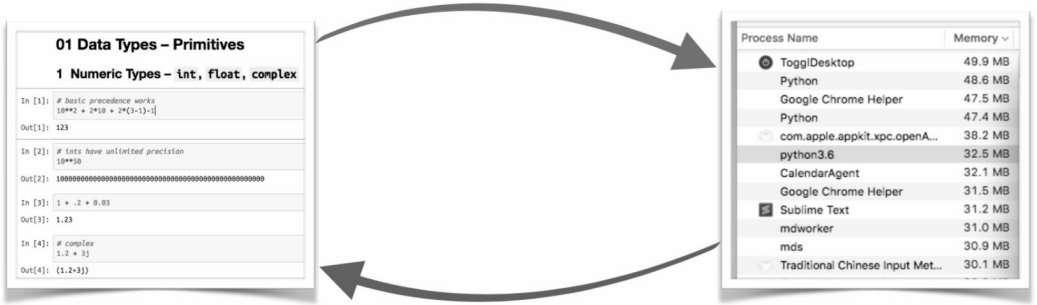


How Jupyter Lab works?

A. When kernel halts or restart,

notebook keeps the outputs.

But when connect to a new kernel,
*kernel remember **nothing**.*



B. "Run All"
to execute all the code again

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

[2]: 12+34

[2]: 46

[3]: print('We are going to rock!!!')
     We are going to rock!!!
```

The Numbers

- In [n]
  - *n* is the execution order, like the line number.
- It may be:
  - 1, 2, 3, 4
  - 3, 5, 2, 4
  - Depends on how you run it.
- "Restart & Run All" to reorder the numbers.

What is an Expression?

Diving into the heart of Python!

Expression: A data value or set of operations to compute a value.

Examples: 1 + 4 * 3

42

Arithmetic operators we will use:

- + - * / addition, subtraction/negation, multiplication, division
- % modulus, a.k.a. remainder
- ** exponentiation

Precedence: Order in which operations are computed.

- * / % ** have a higher precedence than + -
  1 + 3 * 4 is 13
- Parentheses can be used to force a certain order of evaluation.
  (1 + 3) * 4 is 16

I'm learning Python!

Common Math Commands

Python has useful commands for performing calculations!

Command name	Description
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1***`, `***value2***`)`	larger of two values
`min(`***value1***`, `***value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

Constant	Description
`e`	2.7182818...
`pi`	3.1415926...

I'm learning Python!

Variables

Variable is a named piece of memory that can store a value!

- Usage:
  - Compute an expression's result,
  - store that result into a variable,
  - and use that variable later in the program.

Assignment statement: Stores a value into a variable.

- Syntax:

  name = *value*

- Examples: x = 5

  gpa = 3.14

- A variable that has been given a value can be used in expressions.

  x + 4 is 9

I'm learning Python!

Basic Data Types

Many datatypes to choose from in Python!

Integers (default for numbers)

```
z = 5 / 2  # Answer 2, integer division
```

Floats

```
x = 3.456
```

Strings

Can use “” or ‘’ to specify with “abc” == ‘abc’
Unmatched can occur within the string: “matt’s”
Use triple double-quotes for multi-line strings or strings than contain both ‘ and “ inside of them:
“““a‘b“c”””

I’m learning Python!

Significance of Whitespace in Python

Whitespace is meaningful in Python: Especially indentation and placement of newlines

Use a newline to end a line of code

Use \ when must go to next line prematurely

No braces {} to mark blocks of code, use *consistent* indentation instead

- First line with *less* indentation is outside of the block
- First line with *more* indentation starts a nested block

Colons start of a new block in many constructs, e.g. function definitions, then clauses.

I'm learning Python!

Comments

Comments help us understand the code better!

Start comments with #, rest of line is ignored

Can include a “documentation string” as the first line of a new function or class you define

Development environments, debugger, and other tools use it: it’s good style to include one.

#ImAComment

I’m learning Python!

Comments

So how do you put some value in a variable?

Binding a variable in Python means setting a *name* to hold a *reference* to some *object*

Assignment creates references, not copies!

Names in Python do not have an intrinsic type, objects have types

Python determines the type of the reference automatically based on what data is assigned to it

You create a name the first time it appears on the left side of an assignment expression:

x = 3

A reference is deleted via garbage collection after any names bound to it have passed out of scope.

Python uses *reference semantics* (more later)

I'm learning Python!

Naming Rules

Watch out for reserved words!

Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.

```
bob Bob _bob _2_bob_ bob_2 BoB
```

There are some reserved words:

and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while

I'm learning Python!

Naming Convention

We have 3 recommended conventions

The Python community has these recommended naming conventions

- joined_lower for functions, methods and, attributes!
- joined_lower or ALL_CAPS for constants!
- StudlyCaps for classes!

Attributes: interface, _internal, __private

I'm learning Python!

Assignment - An easier way?

Less of typing means more of coding!

You can assign to multiple names at the same time

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
```

This makes it easy to swap values

```
>>> x, y = y, x
```

Assignments can be chained

```
>>> a = b = x = 2
```

I'm learning Python!

Accessing Non Existent Names

Let's see how we can raise an error!

Accessing a name before it's been properly created (by placing it on the left side of an assignment), raises an error

```
>>> y

Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
    y
NameError: name 'y' is not defined
>>> y = 3
>>> y
3
```

I'm learning Python!

Casting

We can explicitly convert based on requirement

Casting in python is therefore done using constructor functions:

•int() - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)

•float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)

•str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

I'm learning Python!

Casting

We can explicitly convert based on requirement

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
```

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

I'm learning Python!

Arithmetic Operators

We can perform common math operations easily!

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x – y
*	Multiplication	x * y
/	Division	x / y
*	Modulus	x % y
**	Exponentiation	x ** y
//	Floor Division	x // y

I'm learning Python!

Comparison Operators

We can compare two values easily!

Operator	Name	Example
==	Equal	x == y
!=	Not Equal	x != y
>	Greater Than	x > y
<	Less Than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

I'm learning Python!

Logical Operators

Can we combine conditional statements?
Yes, we can!

Operator	Description	Example
And	Returns True if both statements are true	x < 5 and x < 10
Or	Returns True if one of the statements is true	x < 5 or x < 4
Not	Reverse the result, returns False if the result is True	not(x , 5 and x < 10)

I'm learning Python!

Identity Operators

Compare objects, not to check if they are equal but to check if they are the same object

Operator	Description	Example
is	Returns True if both variables are same object	x is y
is not	Returns True if both variables are not same object	x is not y

I'm learning Python!

Bitwise Operators

Let's compare binary numbers and operate on them!

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits are 1
^	XOR	Sets each bit to 1 if only one of two bits in 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of leftmost bit in from the left and let the rightmost bits fall off

I'm learning Python!

List Methods

Python has a set of built-in methods that you can use!

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

I'm learning Python!

Tuple Methods

Python has two built-in methods you can use on tuples!

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

I'm learning Python!

Set Methods

Python has two built-in methods you can use on tuples!

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another specified set
discard()	Remove the specified item
intersection()	Returns the intersection of two other set
issubset()	Returns whether another set contains this set or not
pop()	Removes an element from the set

I'm learning Python!

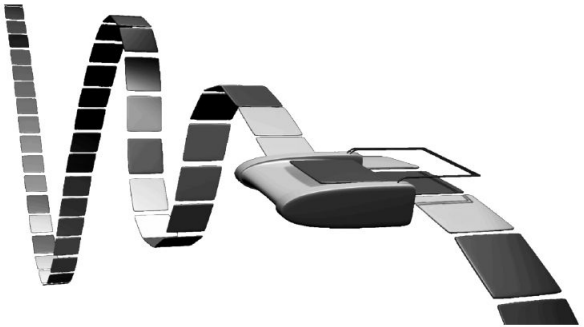
Dictionary Methods

Python has a set of built-in methods you can use on dictionaries

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with specified keys and values
get()	Returns the value of the specified key
items()	Returns a list containing the tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
update()	Updates the dictionary with specified key-valu pairs
pop()	Removes an element from the dictionary

I'm learning Python!

If & While



- `if <condition>: ...`
  - Run inner block *if* true.
- `while <condition>: ...`
  - The while-loop.
  - Run inner block *while* true.
  - I.e. run until false.

For & Loop Control Statements

- `for`
  - The for-loop.
  - In each iteration, get the next item of a collection.
  - Supports str, list, tuple, set, and dict, etc.
  - I.e. iterate an iterable.
- `break`
  - Leave the loop.
- `continue`
  - Go the next iteration.
- `loop … else`
  - If no break happens, execute the *else*.

Pass

- `pass`
  - Do nothing.
  - The compound statements must have one statement.
    - The if, while, for, etc.

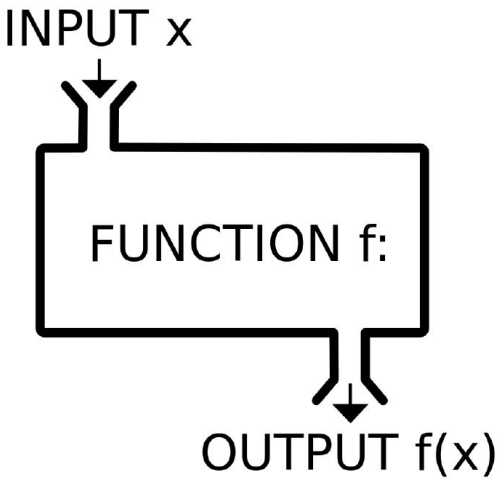
Raise Your Exception

- `raise RuntimeError('should not be here')`
  - Raise an customized exception.
  - Use *class* to customize exception class.
- `raise`
  - Re-raise the last exception.

The Guidelines of Using Try

- An exception stops the whole program.
- However sometimes stop is better than a bad output.
- Only catch the exceptions you expect.
- But catch everything before raise to user.
- And transform the exception into log, email, etc.

Functions



- Reuse statements.
- `def`
  - Take the inputs.
- `return`
  - Give an output.
  - If no return, returns *None*.
- Method
  - Is just a function belonging to an object.

Recap

- When calling function:
  - Keyword arguments: *f(a=3.14)*
  - Unpacking argument list: *f(*list_like, **dict_like)*
- When defining function:
  - Default values: *def f(a=None): ...*
  - Arbitrary argument list: *def f(*args, **kwargs): ...*
- Using docstrings to cooperate with others.

Comprehensions & Generator Expression

- List Comprehension []
- Set Comprehension {}
- Dict Comprehension {:}
- Generator Expression ()

The Functional Tricks

- The functional programming:
  - Is a programming paradigm.
  - Avoids changing-state and mutable data.
  - Sometimes makes code clean, sometimes doesn't.
  - Use it wisely.
- Python is *not* a functional language.
- But provides some useful tools.

The Object-Oriented Programming

- Class makes objects.
- Customize your:
  - Data type.
  - And its operations.
- A powerful tool to abstract the world.

The Object-Oriented Terms in Python

object	Everything in Python is an object, e.g., `str`, `'str'`.
class	Makes instances, e.g., `str`.
instance	Made from class, e.g., `'str'`.
attribute	Anything an object owns.
method	A function an object owns.

`str`	object | class
`str.__class__`	object | attribute | class
`str.split`	object | attribute | function | method
`'str'`	object | instance
`'str'.split`	object | attribute | function | method

Duck - Typing & Protocol

- Duck-Typing
  - “If it looks like a duck and quacks like a duck, it must be a duck.”
  - Avoids tests using `type()` or `isinstance()`.
  - Employs `hasattr()` tests or EAFP programming.
  - EAFP: easier to ask for forgiveness than permission.
- Iterator protocol, for example:
  - `__iter__()` returns itself.
  - `__next__()` returns the next element.
  - The for-loops use the iterator protocol to iterate an object.

The Guidelines of Designing Classes

- Don't use class, unless:
  - Many functions have the same arguments, e.g.,:
    - `def create_user(uid, ...): ...`
    - `def retrieve_user(uid, ...): ...`
    - `def update_user(uid, ...): ...`
  - When design or implementation.
- Don't use class method, etc., unless:
  - You're sure the method is only associate with class.