AlphaDevs

AN INITIATIVE OF ALPHA-X

BASICS OF AI WITH PYTHON: REFRENCE MATERIAL

Basic Terminologies!

Computer Languages:

A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.

Each programming language has a unique set of keywords (words that it understands) and a special syntax for organizing program instructions.

A scripting language is a one that supports scripts. Scripts: programs that are written for a special run-time environment that automates the execution of a task.

Types of programming languages:

- Declarative languages.
- Procedural languages.
- Object oriented languages.
- Functional languages.

Programming Basics

- **Program**: A sequence of statements that have been crafted to do something.
- **code or source code:** The sequence of instructions in a program.
- **syntax:** The set of legal structures and commands that can be used in a particular programming language.
- **output:** The messages printed to the user by a program.
- **sequential execution:** Perform statements one after another in the order they are encountered in the script.
- **conditional execution:** Check for certain conditions and then execute or skip a sequence of statements.
- **repeated execution:** Perform some set of statements repeatedly, usually with some variation.
- **reuse:** Write a set of instructions once and give them a name and then reuse those instructions as needed throughout your program.
- **console:** The text box onto which output is printed.
-

Learning a programming Language!

The essential “pieces” necessary for defining and learning any programming language:

- **Syntax**: How do you write the various parts of the language?
- **Semantics**: What do the various language features mean? For example, how are expressions evaluated?
- **Idioms**: What are the common approaches to using the language features to express computations?
- **Libraries**: What has already been written for you? How do you do things you could not do without.
- **library support** (like access files)?
- **Tools**: What is available for manipulating programs in the language (compilers, repl, debuggers, ...)

Python

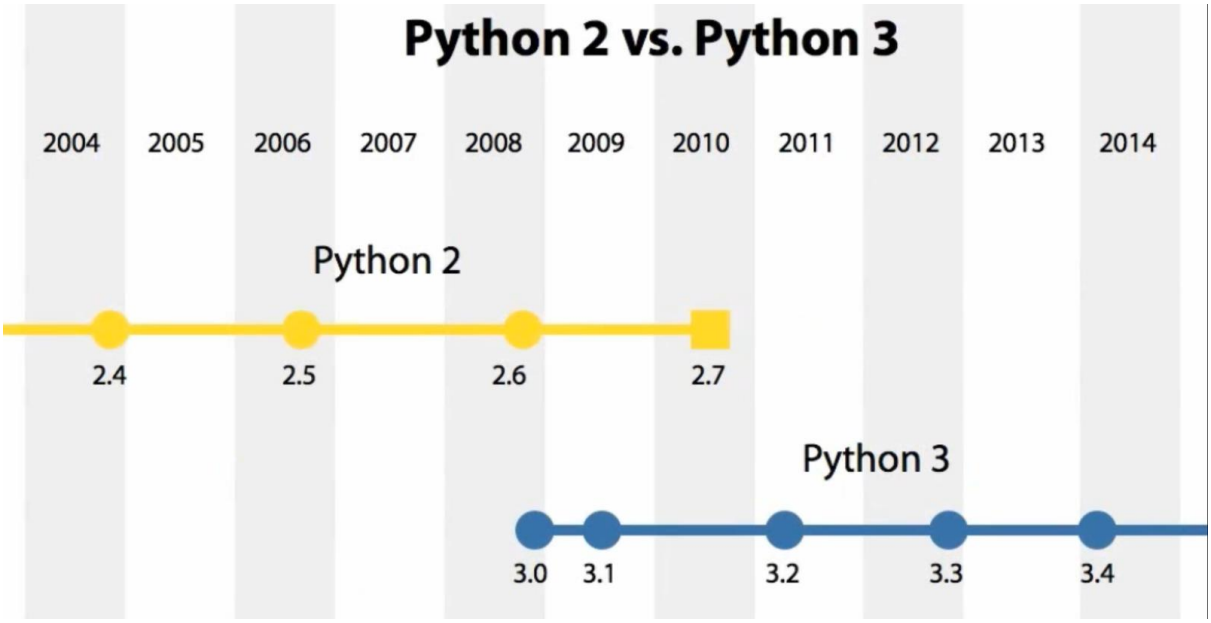
Features of Python:

- Uses an elegant syntax, making the programs you write easier to read.
- Python is clear, readable and expressive.
- Highly readable code makes maintenance of code easy.
- This makes Python ideal for prototype development without compromising maintainability.
- Comes with a large standard library that supports many common programming tasks.
- Python's interactive mode makes it easy to test short snippets of code.
- Runs anywhere, including Mac OS X, Windows, Linux, and Unix, with unofficial builds also available for Android and iOS.
- Is free software in two senses. It doesn't cost anything to download or use Python, or to include it in your application. Python can also be freely modified and re-distributed, because while the language is copyrighted it's available under an open source license.
- A variety of basic data types and complex data types are available.
- Python supports object-oriented programming with classes.
- Code can be grouped into modules and packages.
- The language supports raising and catching exceptions, resulting in cleaner error handling.
- Data types are strongly and dynamically typed. Statically typed languages do type checking at compile time and give every variable a type before the program can be run. Conversely, dynamically typed languages do type checking at run-time, determining types as the program runs.
- Mixing incompatible types (e.g. attempting to add a string and a number) causes an exception to be raised, so errors are caught sooner.

Brief History of python

- Python 1 - 1994
- Python 2 - 2000
- Python 3 - 2008

Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.



Basics Of Python.

Value:

A *value* is one of the basic things a program works with, like a letter or a number. Example 1, 2, and "Hello, World!".

Variables:

One of the most powerful features of a programming language is the ability to manipulate *variables*. A variable is a name that refers to a value.

Variable assignment using = operator.
Naming your variables keep in mind!!
A type of **identifier**.
Rules for naming identifiers:

- The first character of the identifier must be a letter of the alphabet (uppercase ASCII or lowercase ASCII or Unicode character) or an underscore (_).
- The rest of the identifier name can consist of letters
- Identifier names are case-sensitive. For example, myname and myName are not the same. Note the lowercase n in the former and the uppercase N in the latter.

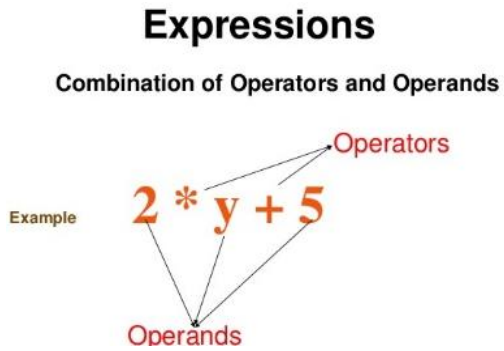
Examples of valid identifier names are i, name_2_3.
Examples of invalid identifier names are 2things, this is spaced out, my-name and >a1b2_c3.

Python reserves 33 keywords:

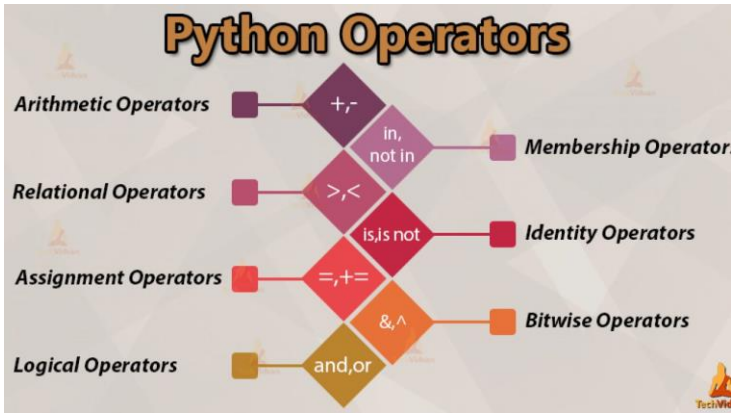
```
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               def                 if                  raise
None                del                 import              return
True                elif                in                  try
and                 else                is                  while
as                  except              lambda              with
assert              finally             nonlocal            yield
break               for                 not
class               from                or
continue            global              pass
```

Expression:

A combination of variables, operators, and values that represents a single result value.



Operators:



We'll be focusing on:
1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operator(s)

Arithmetic Operators

Symbol	Task performed
+	Addition
-	Subtraction
*	Multiplication

/	Division
**	Exponentiation
//	Floored Division

Relational Operators

Symbol	Task performed
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

Logical Operators

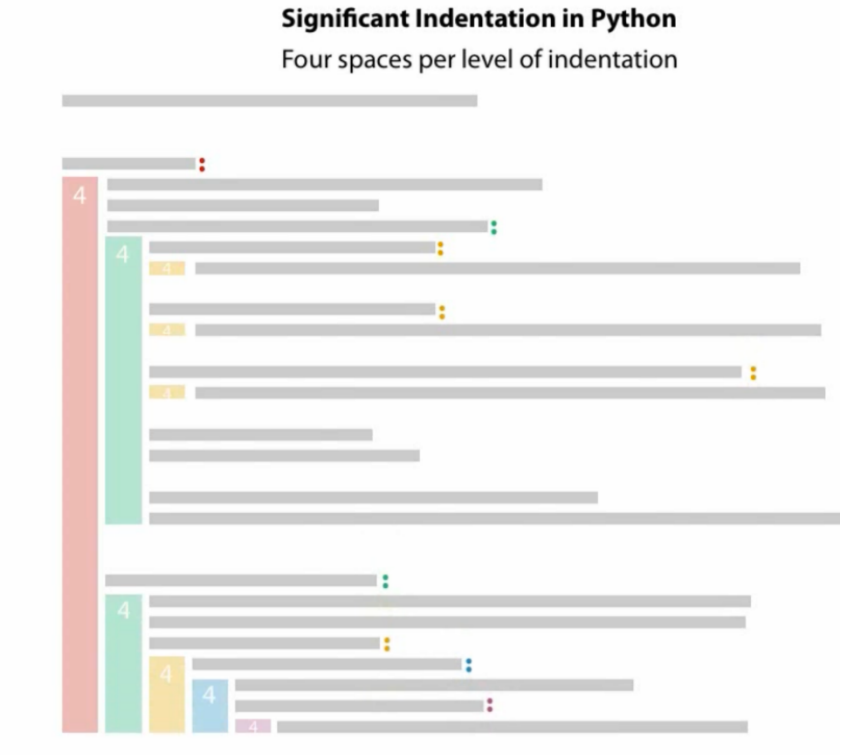
Symbol	Task performed
and	and logic
or	or logic
not	negation

You all must be familiar with BODMAS , in python the concept of evaluation is almost the same, but instead of BODMAS, we use PEMDAS (Parenthesis , Exponentiation, Multiplication, Division, Addition, Subtraction)

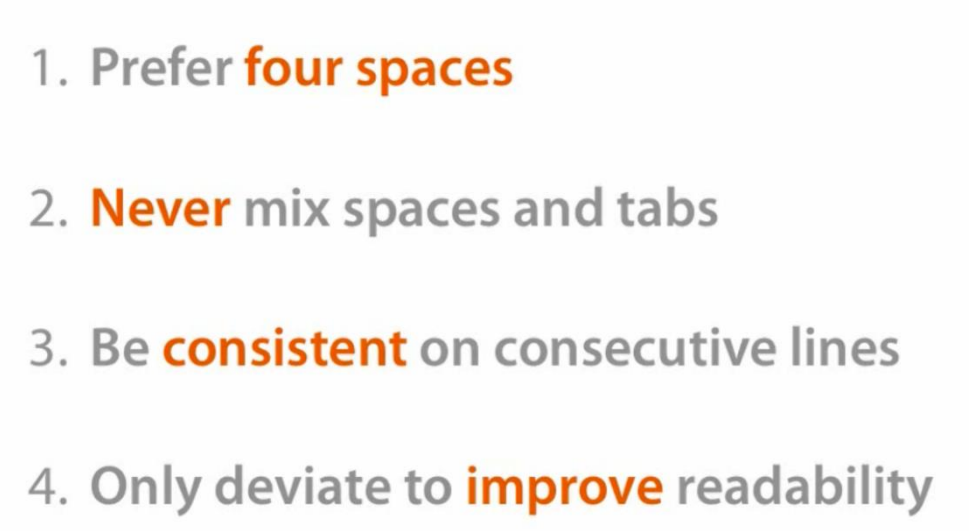
And another important thing to note is that, the order of evaluating an expression is always from LEFT to RIGHT

Indentation in Python

Indentation is a very important concept of Python because without proper indenting the Python code, you will end up seeing IndentationError and the code will not get compiled.



THINGS TO REMEMBER ABOUT INDENTATION



Elements of Flow Control

Flow control statements often start with a part called the *condition*, and all are followed by a block of code called the *clause*.

Conditions

Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

If – Elif - Else

These conditions can be used in several ways, most commonly in "if statements" and loops.
An "if statement" is written by using the `if` keyword.

Example

If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

In this example we use two variables, `a` and `b`, which are used as part of the if statement to test whether `b` is greater than `a`. As `a` is `33`, and `b` is `200`, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

`If statement, without indentation (will raise an error):`

```
a = 33
b = 200
if b > a:
print(" b is greater than a ) # you will get an error
```

Types of Statement

- If statement
- Elif
- Else

EXAMPLE OF ALL STATEMENTS TOGETHER

```
if expression:
    statement1
    statement2
elif expression:
    statement1
    statement2
else:
    statemet1
    statement2
```

LOOPS

In general, statements are executed sequentially:
BUT, using IF-ELSE we can skip part of the code and break the sequence!
Now the question is, what if we want to repeat few statements again and again?
For that we use LOOPS.

Sr.No.	Loop Type & Description

1	while loop
	
	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
	
	`while expression:  # Here as long as the expression is True the statement 1`
	`    statement 1  # and statement 2 will be executed repeatedly`
	`    statement 2`
	`Example:`
	`x = 10`
	`while x > 0:`
	`    print(x)`
	`    x = x - 1`
	
	The while statement is used for repeated execution as long as an expression is true:
	
	This repeatedly tests the expression and, if it is true, executes the first suite; if the expression is false (which may be the first time it is tested) the suite of the else clause, if present, is executed and the loop terminates.

2	for loop
	
	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
	
	`for variable in sequence:  # sequence types like string, range(), list etc.`
	`  statement 1  # the variable will take values in the order of the`
	`  statement 2  # sequence`
	
	`Example:`
	`for i in range (0,10):  # The second arg is not included.`
	`  print(i)  # so it will print from 0 to 9`
	
	For Loop
	
	• The for keyword
	• A variable name
	• The **in** keyword: **this test whether or not a sequence contains a certain value.**
	• A call to the range() method with up to three integers passed to it
	• A colon
	• Starting on the next line, an indented block of code (called the for clause)
	• you can use continue and break statements only inside while and for loops.
	• Range fn accepts 1, 2 or 3 args.

Functions:

A function is a block of code which only runs when it is called.
You can pass data, known as parameters, into a function.
A function can return data as a result.

You're already familiar with the print(), input(), and len() functions. Python provides several built-in functions like these, but you can also write your own functions. A *function* is like a mini-program within a program.

why functions?

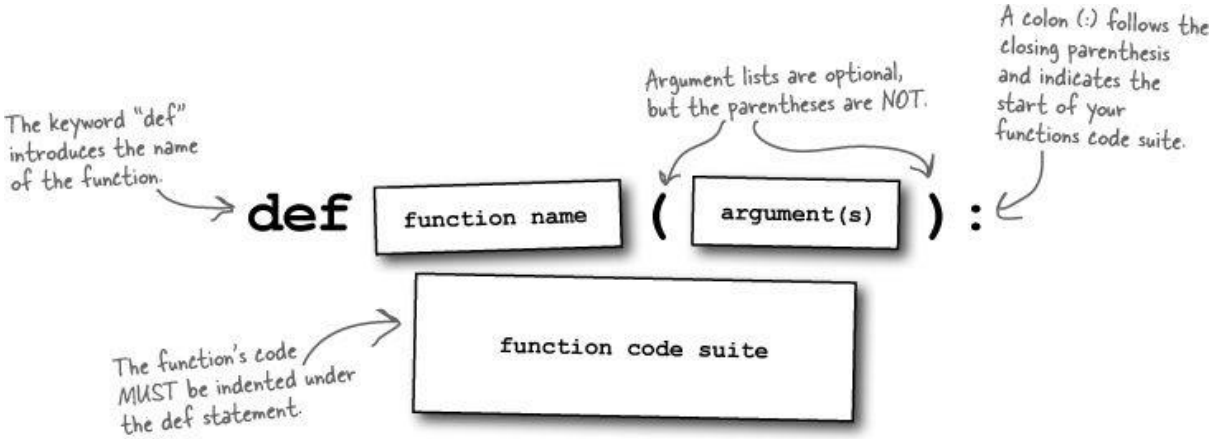
- Creating a new function gives you an opportunity to name a group of statements, which makes your program easier to read, understand, and debug.
- Functions can make a program smaller by eliminating repetitive code. Later, if you make a change, you only have to make it in one place.
- Dividing a long program into functions allows you to debug the parts one at a time and then assemble them into a working whole.
- Well-designed functions are often useful for many programs. Once you write and debug one, you can reuse it.
- Abstraction - Separation of the interface from the implementation.

A major purpose of functions is to group code that gets executed multiple times. Without a function defined, you would have to copy and paste this code each time.

Function characteristics:

- has a name
- has parameters (0 or more)
- has a docstring (help text: optional but recommended)
- has a body
- returns something

The keyword **def** introduces a **function *definition***. It must be followed by the **function name** and the **parenthesized list of formal parameters**. The statements that form the body of the function start at the next line, and must be indented.



Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

```
def my_function(Customer_name):
  print(Customer_name + " , Hello! ")

my_function("Rohan")
my_function("Yash")
my_function("Mohit")
```