Lecture Refrence

Python

Features of Python:

- Uses an elegant syntax, making the programs you write easier to read.
- Is an easy-to-use language that makes it simple to get your program working. This makes Python ideal for prototype development and other ad-hoc programming tasks, without compromising maintainability.
- Comes with a large standard library that supports many common programming tasks such as connecting to web servers, searching text with regular expressions, reading and modifying files.
- Python's interactive mode makes it easy to test short snippets of code. There's also a bundled development environment called IDLE.
- Is easily extended by adding new modules implemented in a compiled language such as C or C++.
- Can also be embedded into an application to provide a programmable interface.
- 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 are available: numbers (floating point, complex, and unlimited-length long integers), strings (both ASCII and Unicode), lists, and dictionaries.
- Python supports object-oriented programming with classes and multiple inheritance.
- 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. Mixing incompatible types (e.g. attempting to add a string and a number) causes an exception to be raised, so errors are caught sooner.
- Python contains advanced programming features such as generators and list comprehensions.
- Python's automatic memory management frees you from having to manually allocate and free memory in your code.

Basic Terminologies!

Computer Languages:

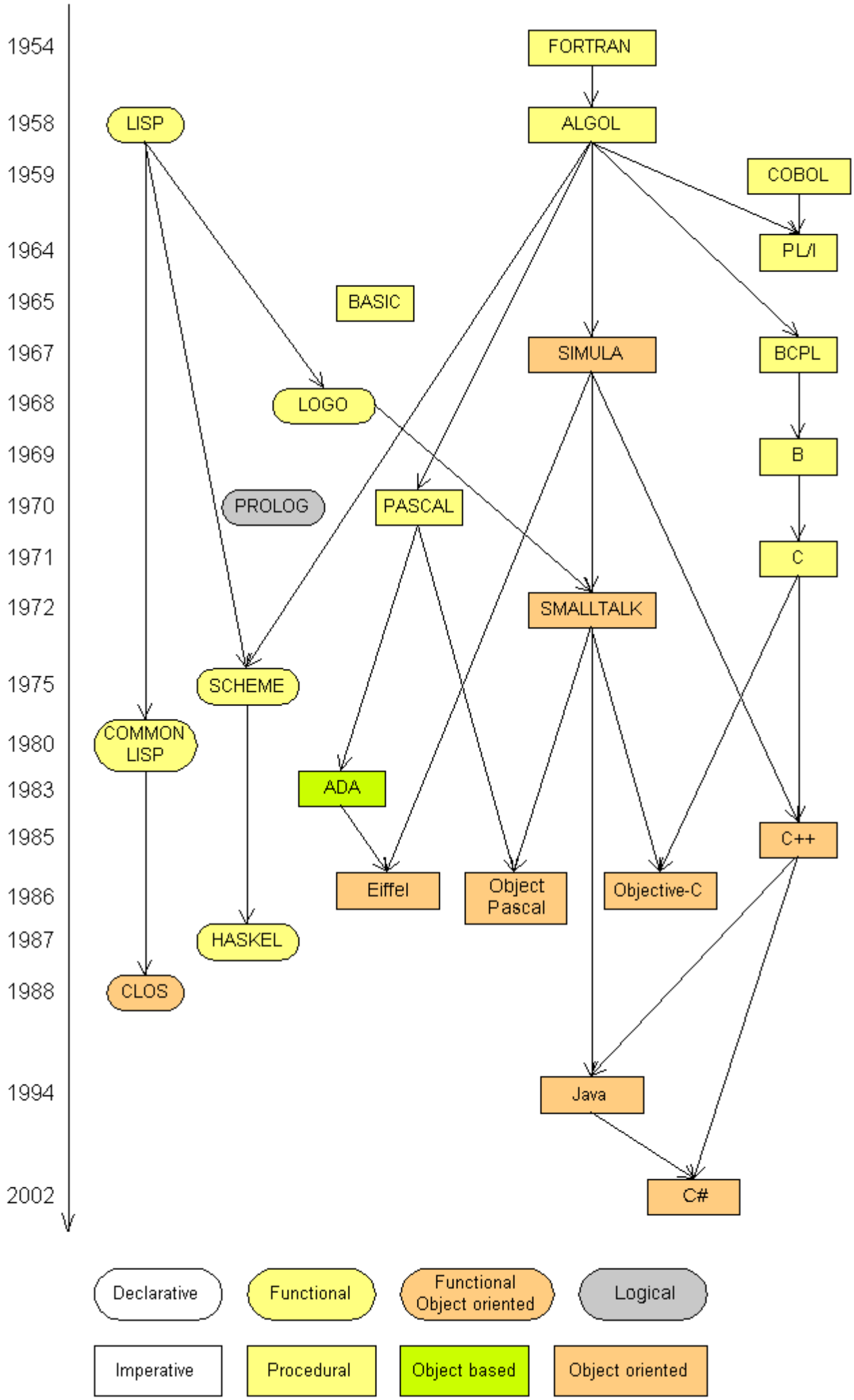
A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.

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

Types:

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


1954
1958
1959
1964
1965
1967
1968
1969
1970
1971
1972
1975
1980
1983
1985
1986
1987
1988
1994
2002
FORTRAN
LISP
ALGOL
COBOL
PL/I
BASIC
SIMULA
BCPL
LOGO
B
PROLOG
PASCAL
C
SMALLTALK
SCHEME
COMMON LISP
ADA
C++
Eiffel
Object Pascal
Objective-C
HASKEL
CLOS
Java
C#
Declarative
Functional
Functional Object oriented
Logical
Imperative
Procedural
Object based
Object oriented

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.

Compiler & Interpreter

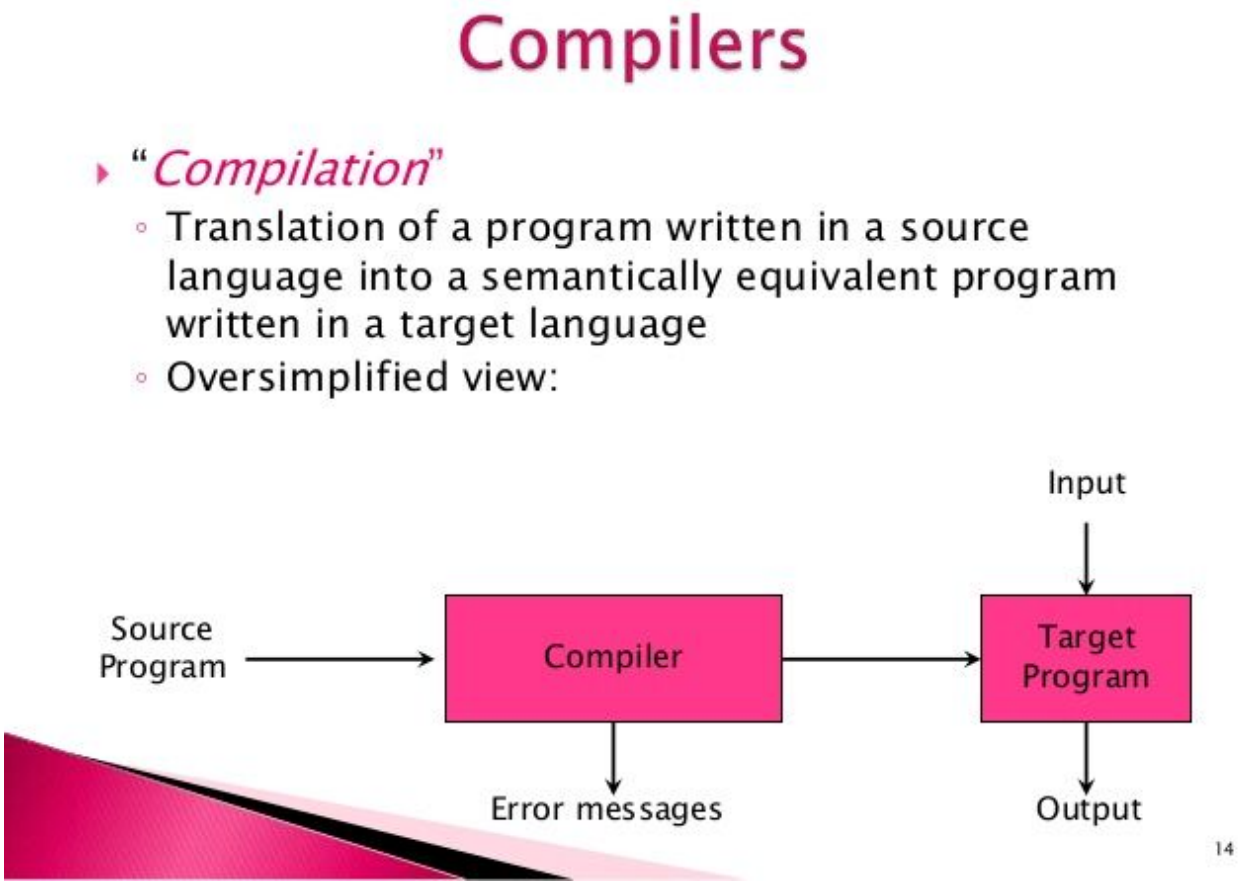


Image result for compiler operations

Compiler and Interpreter are two different ways to execute a program written in a programming or scripting language.

A compiler takes entire program and converts it into object code which is typically stored in a file. The object code is also refereed as binary code and can be directly executed by the machine after linking. Examples of compiled programming languages are C and C++.

An Interpreter directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code. Examples of interpreted languages are Perl, Python and Matlab.

Once a program is compiled, its source code is not useful for running the code. For interpreted programs, the source code is needed to run the program every time.
In general, interpreted programs run slower than the compiled programs.

Python is instead directly *interpreted* into machine instructions.

Behind the scene of compilation!!

- Pre-Processing is the first phase through which source code is passed. This phase include:
  - Removal of Comments
  - Expansion of Macros
  - Expansion of the included files.
- Compiling and produce an intermediate compiled output file which has assembly level instruction.
- In Assembly phase the compiled file is taken as input and turned into an object file by assembler. This file contain machine level instructions. Machine language is tied to the computer hardware, machine language is not *portable* across different types of hardware.
- This is the final phase of linking in which all the linking of function calls with their definitions are done. Linker knows where all these functions are implemented. Linker also adds some extra commands and increases the object file to an executable.
- Through these commands, we know that how output file increases from an object file to an executable file. This is because of the extra code that linker adds with our program.

Install:

- Python3
- Virtualenv
- Jupyter
- IDE of your choice.

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 (uppercase ASCII or lowercase ASCII or Unicode character), underscores (_) or digits (0-9).
- 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:

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

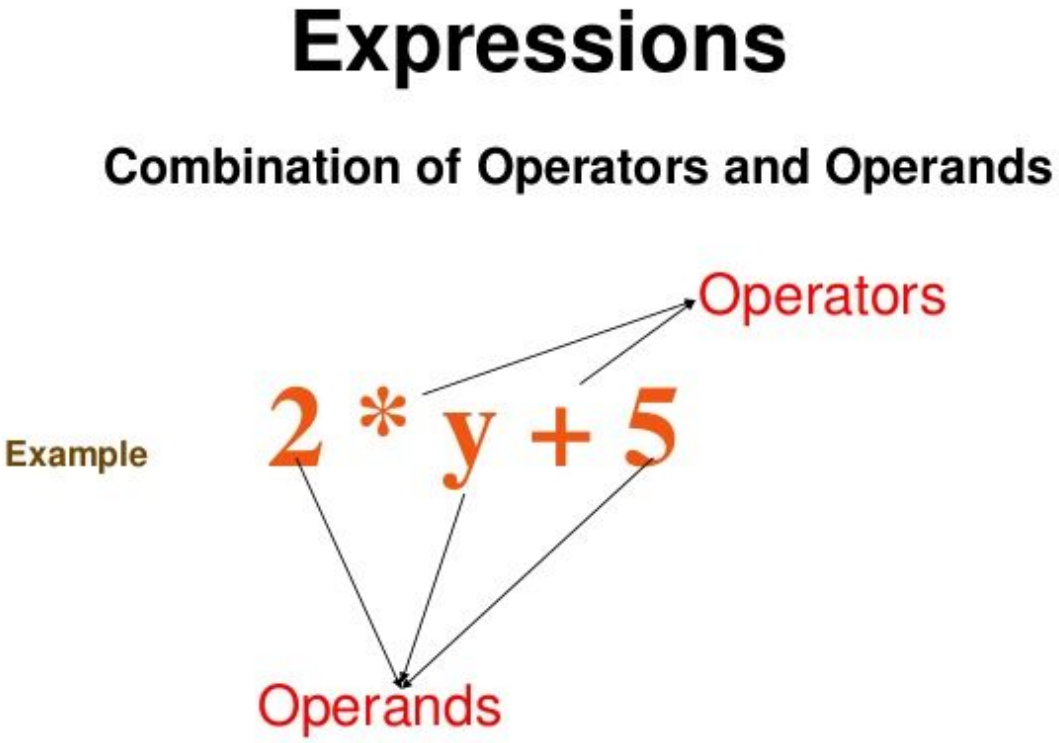
Binding variables and Values.

- Reuse them in the code.

Change binding by reassigning.

Expression:

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



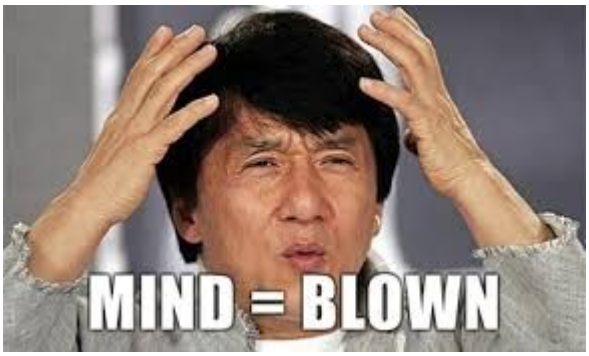
Operators:

Operators are special symbols that represent computations like addition and mul- tiplication. The values the operator is applied to are called *operands*.

Arithmetic Operators

Symbol	Task performed
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
//	Floored Division

Floating point issues



floating point error

Floating-point numbers are represented in computer hardware as base 2 (binary) fractions. For example, the decimal fraction.

```
0.125
```

has value 1/10 + 2/100 + 5/1000, and in the same way the binary fraction

```
0.001
```

has value 0/2 + 0/4 + 1/8. These two fractions have identical values, the only real difference being that the first is written in base 10 fractional notation, and the second in base 2.
Unfortunately, most decimal fractions cannot be represented exactly as binary fractions. A consequence is that, in general, the decimal floating-point numbers you enter are only approximated by the binary floating-point numbers actually stored in the machine.

Stop at any finite number of bits, and you get an approximation. On most machines today, floats are approximated using a binary fraction with the numerator using the first 53 bits starting with the most significant bit and with the denominator as a power of two. In the case of 1/10, the binary fraction is `3602879701896397 / 2 ** 55` which is close to but not exactly equal to the true value of 1/10.

Comparison Operators

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

Boolean Operator

Symbol	Task performed
and	and logic

or	or logic
not	negation

Bitwise Operators

Symbol	Task
|	or
&	and
^	xor
~	negation

When more than one operator appears in an expression, the order of evaluation depends on the **rules of precedence**. For mathematical operators, Python follows mathematical convention. The acronym **PEMDAS** is a useful way to remember the rules:

- **P**arentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 2 * (3-1) is 4, and (1+1)**(5-2) is 8. You can also use parentheses to make an expression easier to read, as in (minute * 100) / 60, even if it doesn't change the result.
- **E**xponentiation has the next highest precedence, so 2**1+1 is 3, not 4, and 3*1**3 is 3, not 27.
- **M**ultiplication and **D**ivision have the same precedence, which is higher than **A**ddition and **S**ubtraction, which also have the same precedence. So 2*3-1 is 5, not 4, and 6+4/2 is 8, not 5.
-
- Operators with the same precedence are evaluated from left to right. So in the expression 5-3-1 is 1, not 3 because the 5-3 happens first and then 1 is subtracted from 2.

Short Circuiting using logical operators.

When Python detects that there is nothing to be gained by evaluating the rest of a logical expression, it stops its evaluation and does not do the computations in the rest of the logical expression. When the evaluation of a logical expression stops because the overall value is already known, it is called **short-circuiting** the evaluation.

Order of Evaluation:

Python evaluates expressions from left to right.
Notice that while evaluating an assignment, the right-hand side is evaluated before the left-hand side.

Built-in Constants:

- False
- True
- None

Comments:

Information in a program that is meant for other programmers (or anyone reading the source code) and has no effect on the execution of the program.

Built-in Data Types:

- int
- float
- string
- Complex

Strings

- The string is enclosed in double quotes if the string contains a single quote and no double quotes, otherwise it is enclosed in single quotes.
- There is no separate character type; a character is simply a string of size one.
- Strings can be indexed (subscripted), with the first character having index 0.
- In addition to indexing, slicing is also supported. While indexing is used to obtain individual characters, slicing allows you to obtain substring.
- Note: The start is always included, and the end always excluded. This makes sure that s[:i] + s[i:] is always equal to s.
- For non-negative indices, the length of a slice is the difference of the indices, if both are within bounds. For example, the length of word[1:3] is 2.
- Out of range slice indexes are handled gracefully when used for slicing.
-
- Python strings cannot be changed — **they are immutable**.

```
+---+---+---+---+---+---+
  P | y | t | h | o | n
+---+---+---+---+---+---+
0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

Mutability And Immutability:

- In object-oriented and functional programming, an **immutable object** (unchangeable object) is an object whose state cannot be modified after it is created. This is in contrast to a **mutable object** (changeable object), which can be modified after it is created.

- Strings and other concrete objects are typically expressed as immutable objects to improve readability and runtime efficiency in object-oriented programming. Immutable objects are also useful because they are inherently **thread-safe**. Other benefits are that they are simpler to understand and reason about and offer higher security than mutable objects.

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Type coercion:

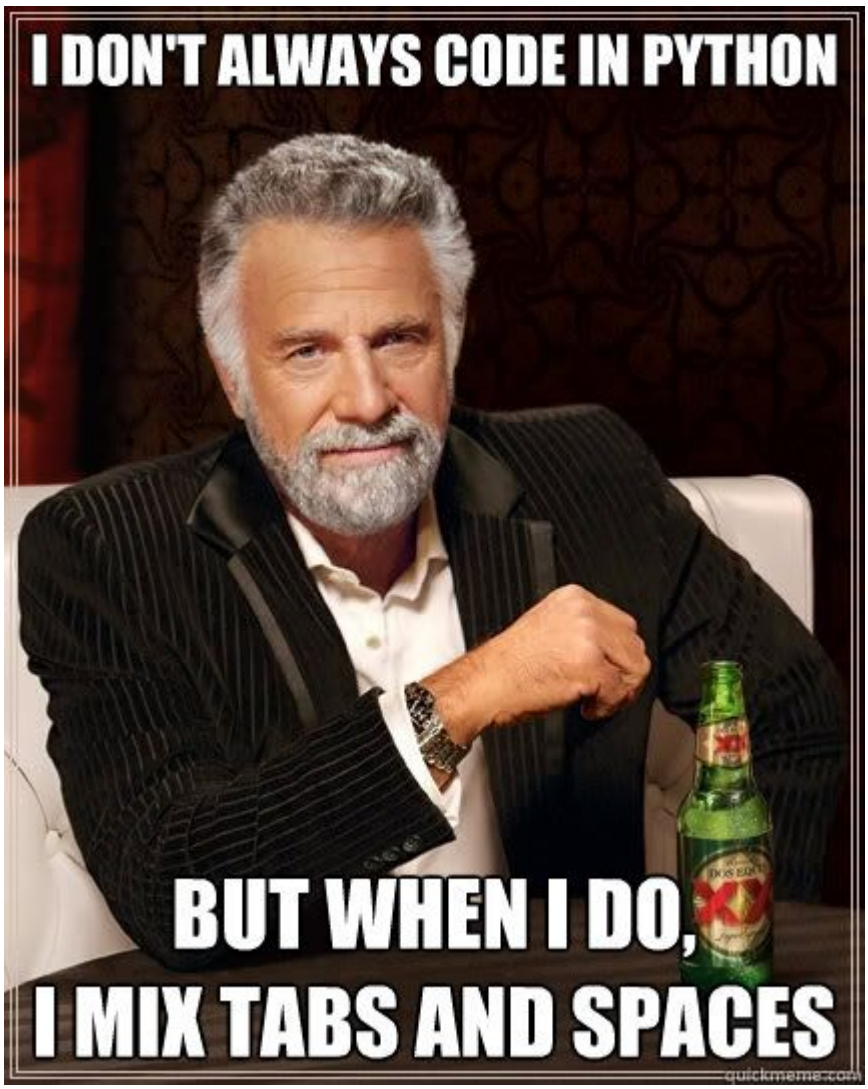
- **Type coercion** is the process of converting value from one type to another (such as string to number, object to boolean, and so on). Any type, be it primitive or an object, is a valid subject for type coercion.
- Implicit
  - If either argument is a complex number, the other is converted to complex;
  - otherwise, if either argument is a floating point number, the other is converted to floating point;
  - otherwise, both must be integers and no conversion is necessary.
- Explicit

Indentation Rules:

indentation

Lines of Python code can be grouped together in blocks. You can tell when a block begins and ends from the indentation of the lines of code. There are three rules for blocks.

1. Blocks begin when the indentation increases.
2. Blocks can contain other blocks.
3. Blocks end when the indentation decreases to zero or to a containing block's indentation.



- Use 4 spaces instead of tabs. Google python style guide.

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

The Boolean expressions you've seen so far could all be considered conditions,
which are the same thing as expressions; *condition* is just a more specific name in the context of flow control statements.
Conditions always evaluate down to a Boolean value, True or False. A flow control statement decides what to do based on whether its condition is True or False, and almost every flow control statement

uses a condition.

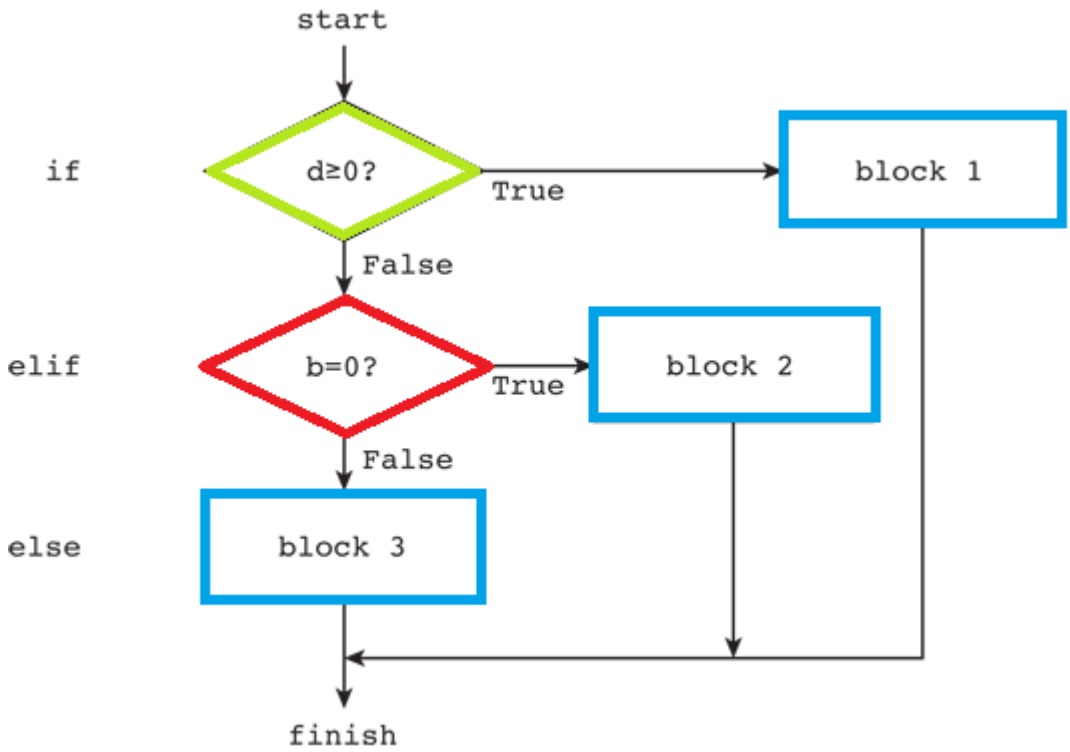
Blocks of Code

Lines of Python code can be grouped together in *blocks*. You can tell when a block begins and ends from the indentation of the lines of code. There are three rules for blocks.

a. Blocks begin when the indentation increases.
b. Blocks can contain other blocks.
c. Blocks end when the indentation decreases to zero or to a containing block's indentation.

A Python program is constructed from code blocks. A *block* is a piece of Python program text that is executed as a unit. The following are blocks: a module, a function body, and a class definition. Each command typed interactively is a block. A script file (a file given as standard input to the interpreter or specified as a command line argument to the interpreter) is a code block.

Conditional Execution



if elif else

- If statement
- Elif
- Else

Nested ladder.

By default, an object is considered true unless its class defines either a __bool__() method that returns False or a __len__() method that returns zero, when called with the object. Here are most of the built-in objects considered false:

- constants defined to be false: None and False.
- zero of any numeric type: 0, 0.0, 0j, Decimal(0), Fraction(0, 1)
- empty sequences and collections: '', (), [], {}, set(), range(0)

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

The Boolean expressions you've seen so far could all be considered conditions,
which are the same thing as expressions; *condition* is just a more specific name in the context of flow control statements.
Conditions always evaluate down to a Boolean value, True or False. A flow control statement decides what to do based on whether its condition is True or False, and almost every flow control statement uses a condition.

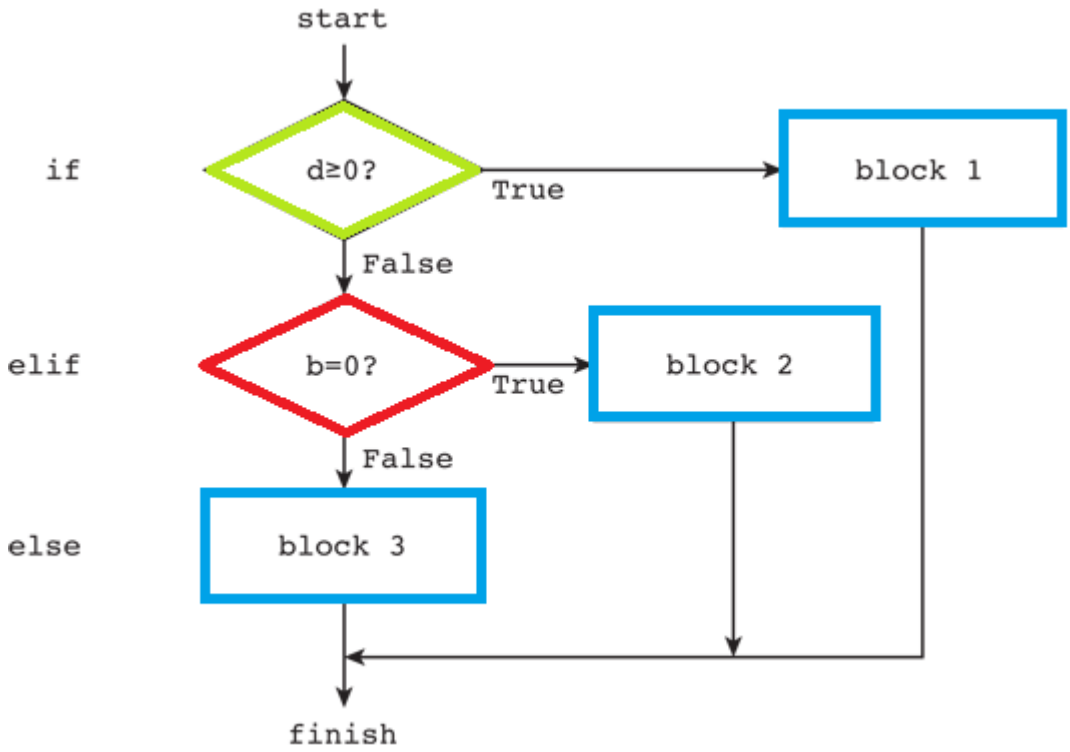
Blocks of Code

Lines of Python code can be grouped together in *blocks*. You can tell when a block begins and ends from the indentation of the lines of code. There are three rules for blocks.

a. Blocks begin when the indentation increases.
b. Blocks can contain other blocks.
c. Blocks end when the indentation decreases to zero or to a containing block's indentation.

A Python program is constructed from code blocks. A *block* is a piece of Python program text that is executed as a unit. The following are blocks: a module, a function body, and a class definition. Each command typed interactively is a block. A script file (a file given as standard input to the interpreter or specified as a command line argument to the interpreter) is a code block.

Conditional Execution



if elif else

- If statement
- Elif
- Else

Nested ladder.

By default, an object is considered true unless its class defines either a __bool__() method that returns False or a __len__() method that returns zero, when called with the object. Here are most of the built-in objects considered false:

- constants defined to be false: None and False.
- zero of any numeric type: 0, 0.0, 0j, Decimal(0), Fraction(0, 1)
- empty sequences and collections: '', (), [], {}, set(), range(0)

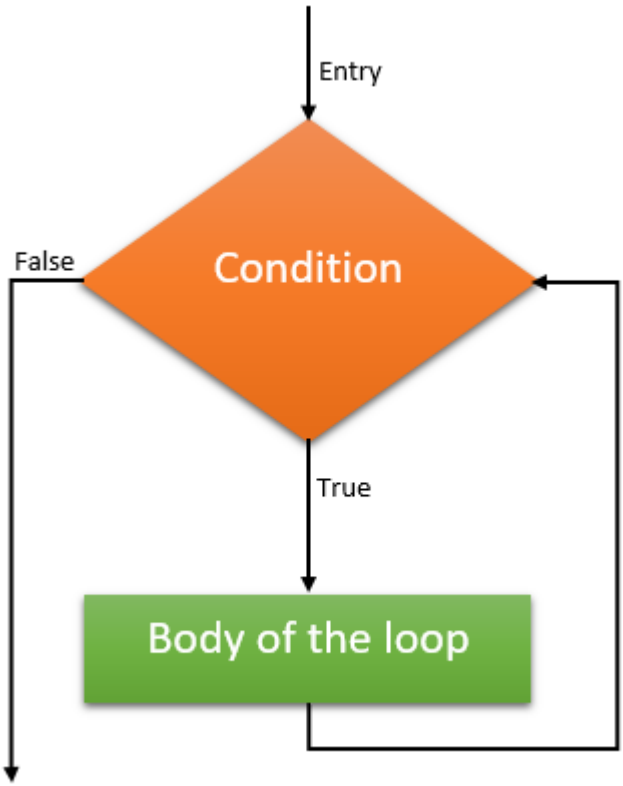
Ternary operator

```
on_true if [expression] else on_false
```

Looping constructs

Repeat your actions using loops!!

While Loop



while loop python

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.
A break statement executed in the first suite terminates the loop without executing the else clause's suite. A continue statement executed in the first suite skips the rest of the suite and goes back to testing the expression.

The for statement in Python differs a bit from what you may be used to in C or C++. Rather than giving the user the ability to define both the iteration step and halting condition (as C), **Python's for statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.**

Range:
If you do need to iterate over a sequence of numbers, the built-in function range() comes in handy. It generates arithmetic progressions.

Break , continue and else statement
The break statement, like in C, breaks out of the innermost enclosing for or while loop.

Loop statements may have an else clause; it is executed when the loop terminates through exhaustion of the loop or when the condition becomes false (with while), but not when the loop is terminated by a break statement.
Pass statement

For Loop

- The for keyword
- A variable name
- The **in** keyword : this tests 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:

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

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.

It's good practice to include docstrings in code that you write, so make a habit of it.

This code is executed when the function is called, not when the function is first defined.

- In code, a function call is just the function's name followed by parentheses, possibly with some number of arguments in between the parentheses.

- When the program execution reaches these calls, it will jump to the top line in the function and begin executing the code there.

- When it reaches the end of the function, the execution returns to the line that called the function and continues moving through the code as before.

why functions?

- Creating a new function gives you an opportunity to name a group of state- ments, 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.

- **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.**

When you call the print() or len() function, you pass in values, called *arguments* in this context, by typing them between the parentheses.

The return statement returns with a value from a function. return without an expression argument returns None.

Common Built-in Functions:

- abs()
- help()
- min()
- max()
- hex() - hexadecimal representation of an integer
- bin() - binary
- oct() - octal
- id()
- input()
- int()
- float()
- str()
- print()
- bool()
- range()
- round()
- pow()
- sum()
- ord()
- len()
- type()

Common str methods:

- capitalize()
- count()
- encode()
- endswith()
- expandtabs()
- find()
- format()
- isalpha()
- isdigit()
- isdecimal()
- islower()
- isupper()
- join()
- startswith()
- swapcase()
- title()

Comparison operator works with strings as well.
Python does not handle uppercase and lowercase letters the same way. All the uppercase letters come before all the lowercase letters.