

Intro to python and programming concepts

BIOINF 606 - Fall 2019

Cristina Mitrea, PhD

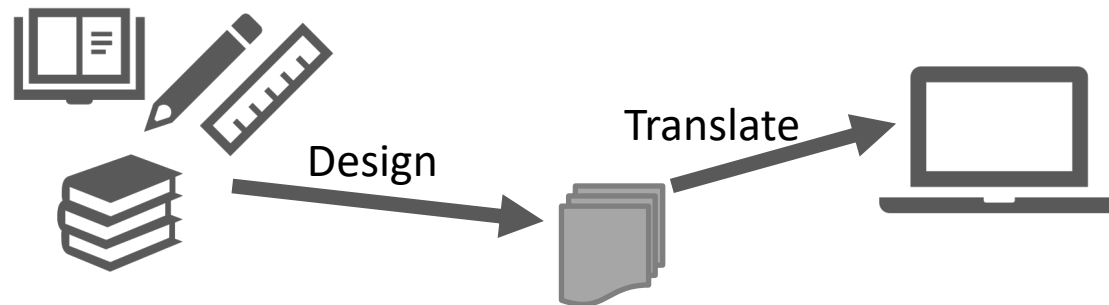
Outline

- What is programming?
- Programming concepts
- What is a programming language?
- What is python?

- What is programming?
 - Definition
 - Problem solving
 - Rules
 - Flowcharts
 - Algorithms
 - Pseudocode
 - Instructions for computer
 - Computer code
 - Binary representation

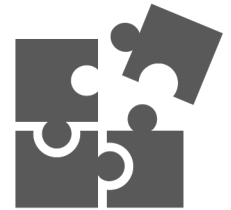
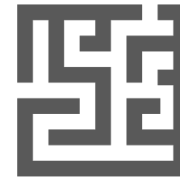
Programming definition

- Programming is the process of designing and building an executable computer program for accomplishing a specific computing task
- Instructing the computer to perform a task
- Coding is translating an algorithm in computer code
- Programming requires problem solving skills



Problem solving

- Problem solving is the process of identifying solutions to a complex problem
- Largely theoretical ... except implementation
 - Define the problem clearly
 - Identify the cause
 - Identify solutions
 - Evaluate and prioritize solutions
 - Implement best alternative
 - Evaluate implementation



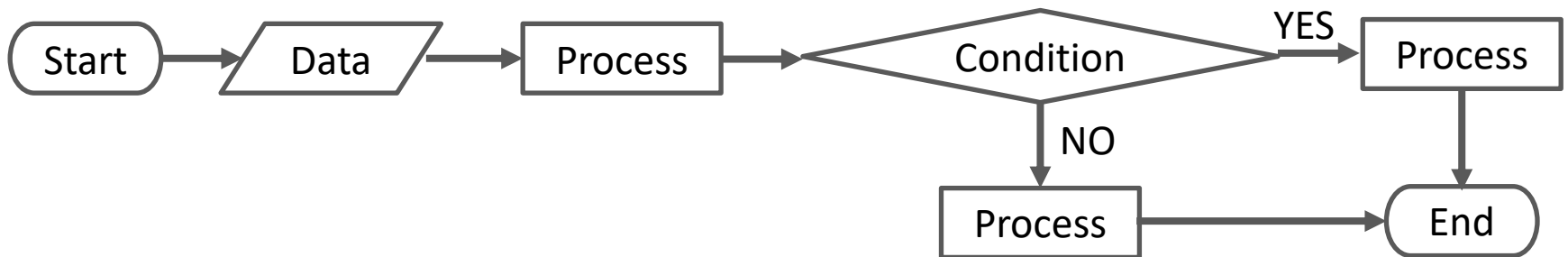
Rules

- In order to solve a problem you need to figure out the rules that restrict or facilitate a solution
- What you have - input
- What you need - result
- Constrains – axioms, formulae, theorems – rules
- Steps to get from the input to the result - algorithm

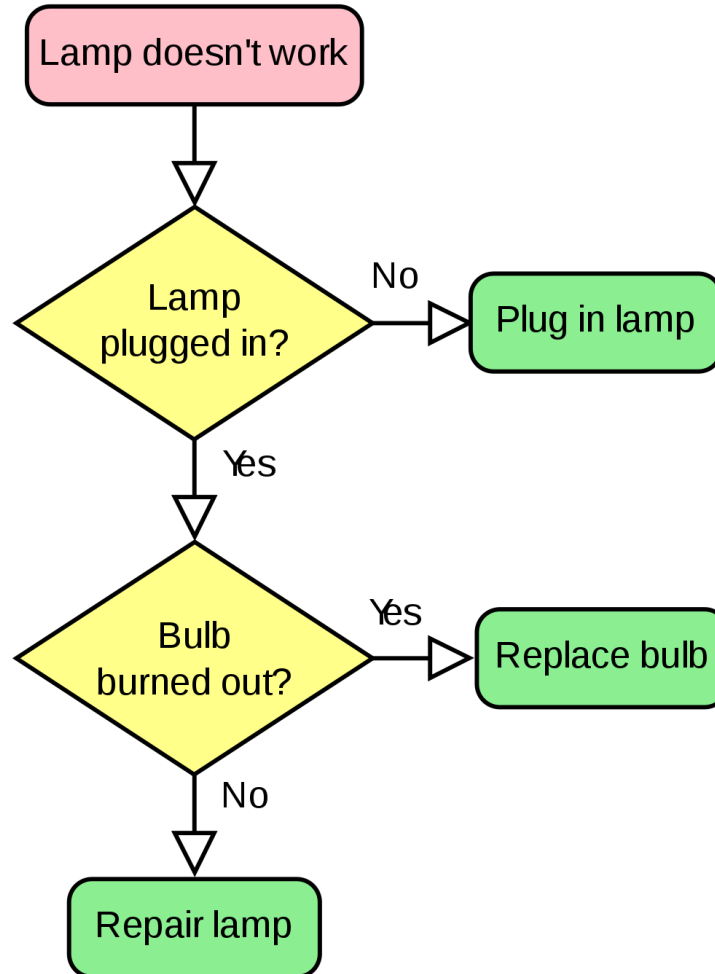


Flowcharts

- A flowchart is a visual representation (diagram) of the sequence of steps and decisions needed to perform a process
 - A diagrammatic representation of an algorithm
 - A step-by-step approach to solving a task



Flowchart example



Algorithms

- An algorithm is a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
 - Step1: Read data
 - Step2: Process data and compute results
 - Step3: Display/Save results

Algorithm example

- For 5 even numbers starting 2 multiply by 3 and then multiply with itself. Display the resulted numbers separated by space.
- Given:
 - Start is 2
 - Count is 5
 - Mfactor is 3
- Result:
 - 36 144 324 576 900
- How do we get from the given data to the result?

Algorithm example

Pseudocode

- Pseudo code is an informal description of an algorithm
 - It is used in program design
 - It can use the structural conventions of a normal programming language
 - It is intended for human reading rather than machine reading

Pseudocode example

1. Start is 2, Count is 5, Mfactor is 3
2. repeat
 1. Counter is 0
 2. Number is Start
 3. Compute is Start * Mfactor
 4. Compute is Compute*Compute
 5. Display Compute
 6. Display ‘ ‘
 7. Number is Number + 2
 8. Counter is Conter + 1
3. until counter = Count

Translate for computer

- Algorithm → refined to Pseudocode
- Pseudocode → refined to very specific instructions
 - Typically done during implementation
- Refined pseudocode – implemented → translated into programming code
- Programming code – interpreter or compiler → translated in to machine code

Computer Languages

Humans



Computers



How can humans “talk to”
(instruct) computers?

Answer: Computer languages (e.g., Java, C, Python, FORTRAN, Basic, C++, Lisp, Ruby, ...)

Programming code/language

- Programming languages are similar to regular written languages
- They are formal languages
- They have vocabulary and syntax
- They evolved based on specific needs
- They rely on and have in common some basic **programming concepts**
- They translate data and instructions to process the data to the computer
 - Data is stored in binary format

Binary representation

- The most used and well known is the base-10 representation
- Binary is a base-2 number system
- Two mutually exclusive states represent information
- States are represented by 0s and 1s

Base-10	Base-2
0	0
1	1
2	10
3	11
4	100

- Programming concepts
 - Variable
 - Data type
 - Scripting
 - Data structure
 - Structured programming
 - Functional programming
 - Object-oriented programming
 - Interpreter
 - Compiler



- A **variable or identifier** is a label or a name given to a location in memory that it refers to which stores a value that is assigned/associated/given to the variable
- The variable can also be interpreted as a container (the location in memory) where a value is stored
- The name:
 - is a combination of letters in lowercase (a to z), uppercase (A to Z), digits (0 to 9), or underscore _
 - cannot start with a digit
 - cannot be a vocabulary/reserved word (keyword)
- A programming language can be case sensitive so variable A and a are not the same (python)
- **Constant** – variable-like: value cannot be changed once initialized
- **Use a meaningful name** – important for code understanding

Variable - rules

- A variable should be **initialized** – given a value – before being used
- A variable is given/assigned a value through an assignment operator
- An **assignment** statement sets and/or re-sets the value stored at the location in memory referred/ denoted by a variable
- In some programming languages a variable also has to be declared
- To **declare** a variable means to bind it to a certain type of value (data type)

Variable - rules

- The **scope** of a variable is the part of the code where the variable is available to use
- The scope is typically given by the place in the code where the variable is declared or initialized
- Inner scope can access variables initialized in an outer scope, but not vice versa
 - Analogy: federal law applies locally, state law does not apply generally
- More about variables:
 - [https://en.wikipedia.org/wiki/Variable_\(computer_science\)](https://en.wikipedia.org/wiki/Variable_(computer_science))

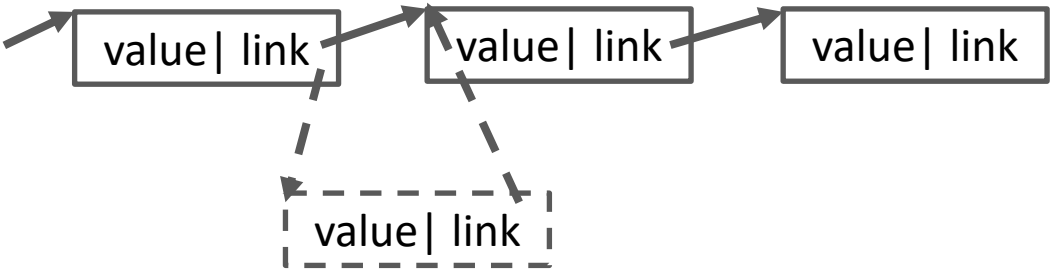
Data type

- Data type is a classification on an attribute of the data that tells the computer how to interpret it, store it and how it can be used (what properties it has, what operators can be applied)
- Data in this context is a category of values
 - e.g. Numeric
- The value of a variable is associated with a data type
 - 0 is Numeric
- They are programming-language specific

Data type

- Basic types:
 - types provided as a building block
 - are typically built-in – built-in support is provided for them
 - python e.g.: Logical (bool), Numeric (int, float, complex), String, List, Tuple, Dictionary
- Classic primitive/basic datatypes: character, integer, floating-point number, boolean, reference (pointer)
- Composite data types or data structures – complex structures derived from basic data types
- More about data types:
 - https://en.wikipedia.org/wiki/Data_type

Data structure

- A data structure is a data organization, management, and storage format that enables efficient access and modification (typically a composite type)
- E.g.: Gene Data:
 - Gene_Symbol string
 - Gene_Expression float
- E.g.: Linked list:
 - Node
 - Value int
 - Link Node
- More about data structures:
 - https://en.wikipedia.org/wiki/Data_structure

Scripting

- Putting together a collection of statements that can be otherwise ran one at a time
- Purpose: automation of tasks
- Scripts are interpreted just like reading a book or following the instructions of a recipe
 - E.g: command line interface (shell) scripting
- Most programming languages can be used for scripting although their main usage is not for that
- More about scripting:
 - https://en.wikipedia.org/wiki/Scripting_language

Structured programming

- Structured or modular programming is a paradigm that proposes features that allow for ease of understanding and modification/maintenance of the code
- It introduces concepts such as control structures (decision/selection, repetition), functions, blocks, and exception handling
- More about structured programming:
 - https://en.wikipedia.org/wiki/Structured_programming

Functional programming

- Functional programming is a paradigm that proposes building the structure of the computer program so that computation is treated as the evaluation of a mathematical function (is a declarative style)
 - https://en.wikipedia.org/wiki/Declarative_programming
- Looping is achieved through recursion
 - A function calling itself until a termination condition is met
 - $F(x) = \begin{cases} x & \text{when } x=10, \\ x+F(x+1), & \text{when } x < 10 \end{cases}$
 - $F(0) = 0+1+2+\dots+9+10 = 45$
- More about functional programming:
 - https://en.wikipedia.org/wiki/Functional_programming

Imperative programming

- Declarative programming is a paradigm
- One example/type of imperative programming is procedural programming – promotes the use of procedures/functions
 - https://en.wikipedia.org/wiki/Procedural_programming
- Machine code (native to the computer, hardware) is imperative
- More about imperative programming
 - https://en.wikipedia.org/wiki/Imperative_programming

Object-oriented programming

- Object oriented programming is a imperative programming paradigm that supports objects
- An object contains data and code
- Data is stored in the object variables also called fields or attributes
- Code is stored in the object methods, which are functions that can manage and process the data
- To be able to create multiple objects of the same type we need a data structure that also has meaningful functions – a class

Classes

- Class – defines the structure and functions for the type of an object
- Objects – class instances – well defined all variables have values, functions are implemented
- E.g. Gene {EntrezId string, Symbol string, ExprValue float}
GeneData {Gene gene, MeasuredValue float, isOverExpressed()}
- To be able for the computer to understand this high-level type of code it need to be transformed into machine code which is done by an interpreter or a compiler
- More about object oriented programming
 - https://en.wikipedia.org/wiki/Object-oriented_programming

Interpreter

- An interpreter is a program that executes the high-level code as soon as it is given to it
- The program always needs the interpreter to run
- Need of different interpreters for different platforms
- More about interpreters:
 - [https://en.wikipedia.org/wiki/Interpreter_\(computing\)](https://en.wikipedia.org/wiki/Interpreter_(computing))

Compiler

- The compiler is a program that transforms source code in machine-level language
- Generates a stand-alone program that can be ran
- The stand-alone program is called an executable
- A compiler could do:
 - parsing, lexical analysis, semantic analysis, conversion to an intermediate representation, code optimization and code generation
- More about compilers:
 - <https://en.wikipedia.org/wiki/Compiler>

- What is a programming language?
 - Definition
 - How programming languages came to be
 - Low-level programming language
 - High-level programming languages

Definition

- A programming language is a formal language used to write computer programs
 - Typically used to implement algorithms
- A formal language contains words with letters from an alphabet that are well-formed according to a set of rules

Formal Language

- A formal language is composed of syntax and semantics
 - syntax – describes, in the form of a context free grammar, the possible combinations of symbols that form a syntactically correct program
 - semantics – defines restrictions on the structure of valid texts (e.g. declare a variable before use)

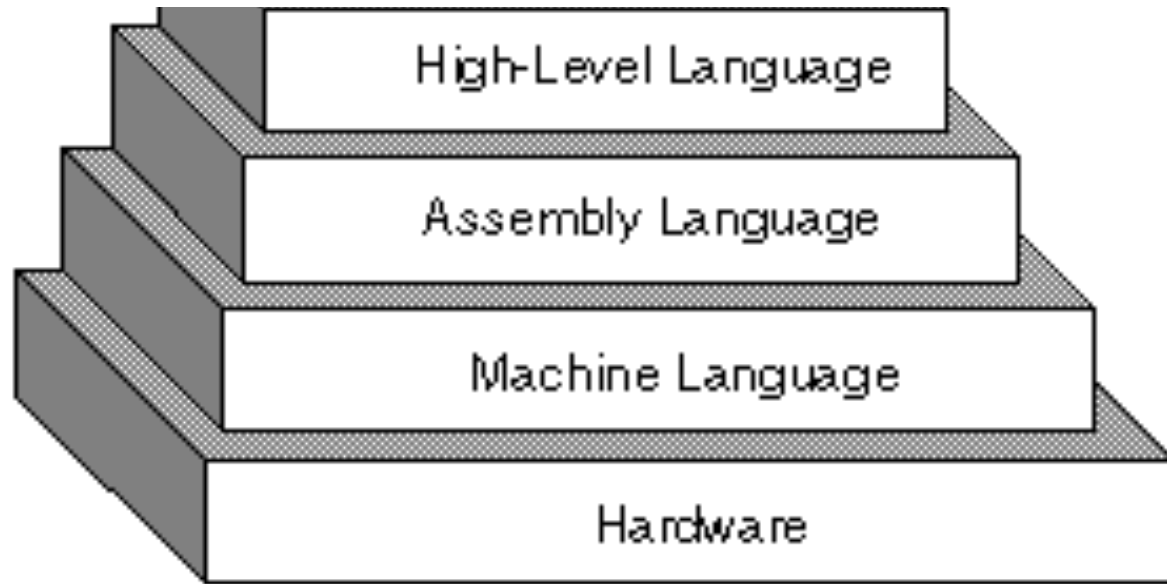
Grammar

- Grammar example:
 - $\langle \text{exp} \rangle ::= \langle \text{exp} \rangle "+" \langle \text{exp} \rangle$
 - $\langle \text{exp} \rangle ::= \langle \text{exp} \rangle "*" \langle \text{exp} \rangle$
 - $\langle \text{exp} \rangle ::= "(" \langle \text{exp} \rangle ")"$
 - $\langle \text{exp} \rangle ::= "a"$
 - $\langle \text{exp} \rangle ::= "b"$
 - $\langle \text{exp} \rangle ::= "c"$
 - https://en.wikibooks.org/wiki/Introduction_to_Programming_Languages/Grammars
- More about programming languages:
 - https://en.wikipedia.org/wiki/Programming_language

How programming languages came to be

- Digital devices only understand binary and the machine circuit patterns which was used by primitive binary languages
- 1949 assembly language was created
- In early compilers, due to limited computer memory compilation was split into several small programs
- Technological advancement led to more resources that allowed for better compiler design and faster compilation

<https://en.wikipedia.org/wiki/Compiler>



[http://wiki.sjs.org/wiki/index.php/History of Computers - Assembly Language](http://wiki.sjs.org/wiki/index.php/History_of_Computers_-_Assembly_Language)

Assembly language

- The microprocessor in a computer manages the computer's arithmetical, logical, and control activities
- It has its own set of machine language instructions (sequences of 0s and 1s) for operations
 - getting input from keyboard
 - displaying information on screen
- https://www.tutorialspoint.com/assembly_programming/assembly_introduction.htm
- Assembly language was designed to make machine language easier to write – it has a more understandable form a very strong correspondence to machine language
- Assembly code is translated into machine code by a program called assembler
 - https://en.wikipedia.org/wiki/Assembly_language

Assembly advantages

- Requires less memory and execution time
- Suitable for time-critical tasks
- Capability for complex hardware-specific tasks
- Most suitable for writing code that deals with communication with external devices

https://www.tutorialspoint.com/assembly_programming/assembly_introduction.htm

Low-level programming language

- Provides little or no abstraction of programming
- Is simple but not clear
- Has a steep learning curve
- E.g. Machine language and Assembly language
- Can be translated to machine code without an interpreter or compiler
- Assembly has a processor for that – assembler
- Allow for total control over the resources
- Requires low resources

https://en.wikipedia.org/wiki/Low-level_programming_language

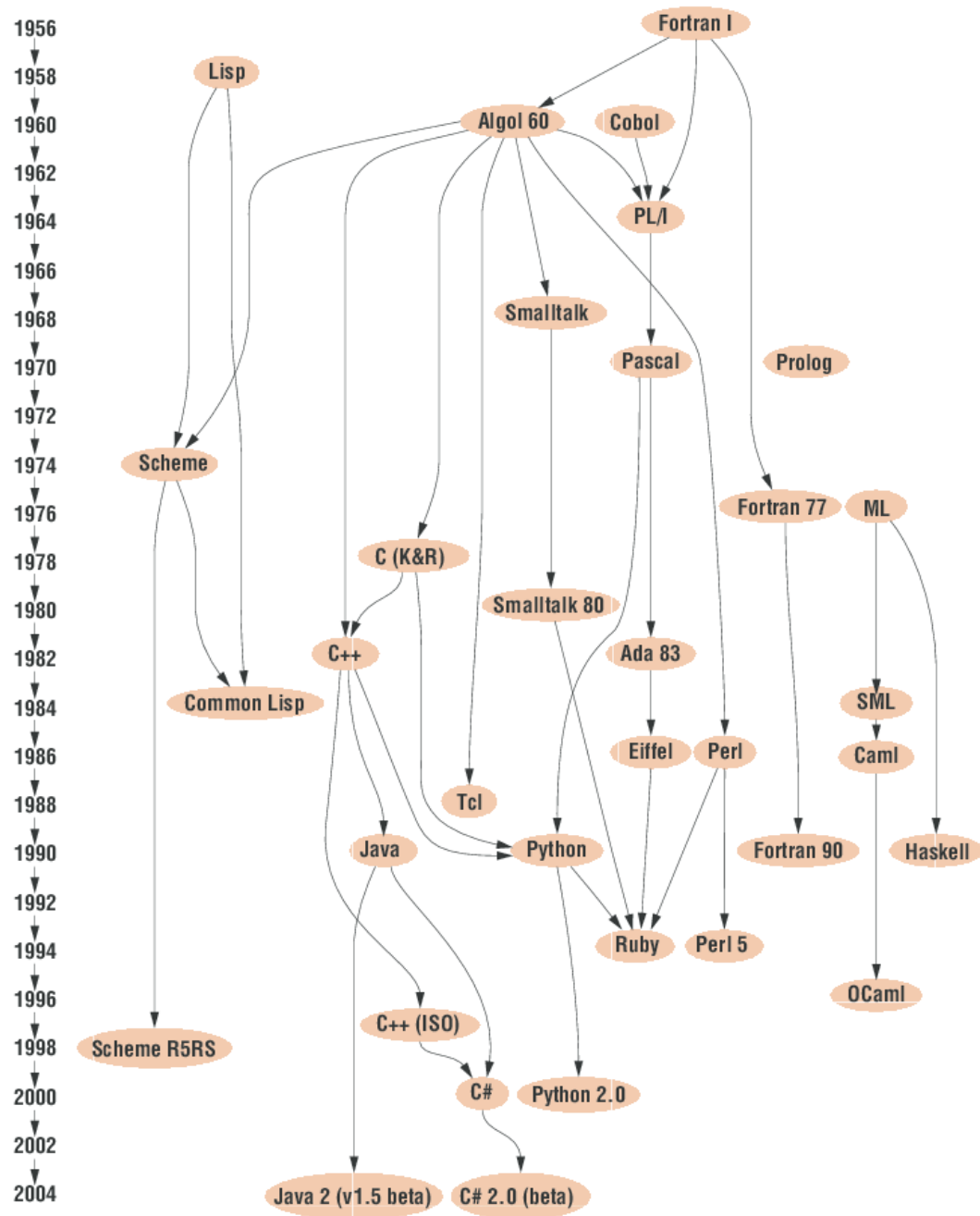
High-level programming language

- Provides a high abstraction of programming
- Is more complex but more clear, closer to human language
- Easier to write, use and maintain
- Portable code
- The first wide-spread high-level language was FORTRAN
 - It was the first commercially available
 - Developed by John Backus at IBM in 1954
 - First FORTRAN manual was available in 1956

https://en.wikipedia.org/wiki/High-level_programming_language

<https://www.computerscience.gcse.guru/theory/high-low-level-languages>

https://en.wikipedia.org/wiki/History_of_programming_languages



Chen, I et al.
IEEE Software
22(3):72- 79
June 2005

- What is python
 - python history
 - python features
 - python specifics
 - Modules
 - Packages
 - Applications

python history



- Guido van Rossum started implementing python in 1989 to bridge the gap between C and Bourne shell for optimization in the Amoeba operating system
- python was developed as a successor to the ABC programming language
- python is named after the late 60's – early 70's British comedy television show Monty python's Flying Circus
- On 2018 Guido van Rossum stepped down as leader (Benevolent dictator for life (BDFL)) in favor of a steering committee of 5

[https://en.wikipedia.org/wiki/python_\(programming_language\)](https://en.wikipedia.org/wiki/python_(programming_language))

python history

- python 0.9.0 - February 20, 1991
 - python 0.9.1 - February, 1991
 - python 0.9.2 - Autumn, 1991
 - python 0.9.4 - December 24, 1991
 - python 0.9.5 - January 2, 1992
 - python 0.9.6 - April 6, 1992
 - python 0.9.8 - January 9, 1993
 - python 0.9.9 - July 29, 1993
- python 1.0 - January 1994
 - python 1.2 - April 10, 1995
 - python 1.3 - October 12, 1995
 - python 1.4 - October 25, 1996
 - python 1.5 - December 31, 1997
 - python 1.6 - September 5, 2000
- python 2.0 - October 16, 2000
 - python 2.1 - April 15, 2001
 - python 2.2 - December 21, 2001
 - python 2.3 - July 29, 2003
 - python 2.4 - November 30, 2004
 - python 2.5 - September 19, 2006
 - python 2.6 - October 1, 2008
 - python 2.7 - July 3, 2010
- python 3.0 - December 3, 2008
 - python 3.1 - June 27, 2009
 - python 3.2 - February 20, 2011
 - python 3.3 - September 29, 2012
 - python 3.4 - March 16, 2014
 - python 3.5 - September 13, 2015
 - python 3.6 - December 23, 2016
 - python 3.7 - June 27, 2018
 - python 3.7.4 - July 8, 2019

python history

- python 2.0 released on 16 October 2000
 - many major new features (list comprehensions)
 - including a cycle-detecting garbage collector and support for Unicode
- python 3.0 released on 3 December 2008. It was a
 - major revision of the language to deal with redundancy
 - not completely backward-compatible
 - many of its major features were backported to python 2.6.x and 2.7.x (the last 2 of the 2.x releases)
 - releases of python 3 include the 2to3 utility - automates (partially) the of python 2 to python 3 code translation

[https://en.wikipedia.org/wiki/python_\(programming_language\)](https://en.wikipedia.org/wiki/python_(programming_language))

python history

- python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that
 - Concern: a large body of existing code could not easily be forward-ported to python 3
- In January 2017, Google announced work on a python 2.7 to Go transcompiler to improve performance

[https://en.wikipedia.org/wiki/python_\(programming_language\)](https://en.wikipedia.org/wiki/python_(programming_language))

python philosophy

- Some of the principles
 - Beautiful is better than ugly
 - Explicit is better than implicit
 - Simple is better than complex
 - Complex is better than complicated
 - Readability counts
 - In the face of ambiguity, refuse the temptation to guess
 - Now is better than never.

https://en.wikipedia.org/wiki/Zen_of_python

python features

- Easy to learn and to use
- Expressive
- Free and Open Source
- Object-Oriented Language
- Supports Graphical Users interfaces Programming
 - GUIs can be made using PyQt5, PyQt4, wxpython or Tk modules
 - PyQt5 – the most popular

<https://data-flair.training/blogs/features-of-python/>

<https://www.geeksforgeeks.org/python-features/>

python features

- High-Level
- Extensible
 - we can write our some python code into c or c++ language and also we can compile that code in c/c++ language.
- Portable
- Integrated
 - we can easily integrated python with other language like c, c++ etc.
- Interpreted
- Large Standard Library
 - provides rich set of module and functions you can use
- Dynamically Typed

<https://data-flair.training/blogs/features-of-python/>
<https://www.geeksforgeeks.org/python-features/>

python features

- High-Level
- Extensible
 - we can write our some python code into c or c++ language and also we can compile that code in c/c++ language.
- Portable
- Integrated
 - we can easily integrated python with other language like c, c++ etc.
- Interpreted

<https://data-flair.training/blogs/features-of-python/>
<https://www.geeksforgeeks.org/python-features/>

python features

- Supports Database Connectivity
 - provides interface to many types of database
- Large Standard Library
 - provides rich set of module and functions you can use
- Dynamically Typed
 - variables are not declared type is inferred from value
- Strongly Typed
 - cannot perform operations between

<https://data-flair.training/blogs/features-of-python/>

<https://www.geeksforgeeks.org/python-features/>

python specifics

- Blocks of code are denoted by line indentation
 - strictly enforced
- Statements should be written on a single line
 - \ is a line continuator – used in multi-line statements
- ; allows for multiple statements on the same line
- Single ' ' double " " and triple ''' ', """ """ quotes
 - Used for string literals (text values)
 - Triples quotes are for text spanning multiple lines
- # starts a comment if not in a string
- """ (docstrings) also starts a comment that spans multiple lines

https://www.tutorialspoint.com/python/python_basic_syntax.htm

python specifics

- Suite – group of individual statements that make a block
- Compound or complex statements require a header line and a suite
 - Header line starts with a keyword and ends with :
 - Keywords: if, while, def, and class
- Command line arguments

https://www.tutorialspoint.com/python/python_basic_syntax.htm

python specifics

- In python Everything is an object
 - variables are references – names for objects
 - the reference has no type
 - the object has type
- In python we can do multiple assignments
 - A value can be given to multiple variables at the same time
- None is used as a non value (null value)

https://www.tutorialspoint.com/python/python_basic_syntax.htm

python keywords

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

Arithmetic operators

Operator	Meaning	Description	Example
+	Addition	Adds values on either side of the operator.	$a + b = 30$
-	Subtraction	Subtracts right hand operand from left hand operand.	$a - b = -10$
*	Multiplication	Multiplies values on either side of the operator	$a * b = 200$
/	Division	Divides left hand operand by right hand operand	$b / a = 2$
%	Modulus	Divides left hand operand by right hand operand and returns remainder	$b \% a = 0$
**	Exponent	Performs exponential (power) calculation on operators	$a ** b = 10 \text{ to the power } 20$
//	Floor Division	The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) –	$9 // 2 = 4$ and $9.0 // 2.0 = 4.0$, $-11 // 3 = -4$, $-11.0 // 3 = -4.0$

Comparison operators

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. Similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Logical operators

Operator	Meaning	Description	Example
and	Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or	Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not	Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

https://www.tutorialspoint.com/python/python_basic_operators.htm

Modules

- Modules are additional pieces of code that further extend python's functionality
- A module typically has a specific function
 - additional math functions, databases, network...
- A module is a file that contains python code
 - typically the file is: `module_name.py`
- python comes with many useful modules
- To bring the module functionality in your code
 - `import module_name`
 - `from module_name import function_name`

Packages

- Packages are directories with an **`__init__.py`** file
 - the file can be empty or contain instructions to restrict which of its module can be imported by code outside the package
 - contain packages and modules
 - to import a module from a package use:
 - `import package_name.module_name`

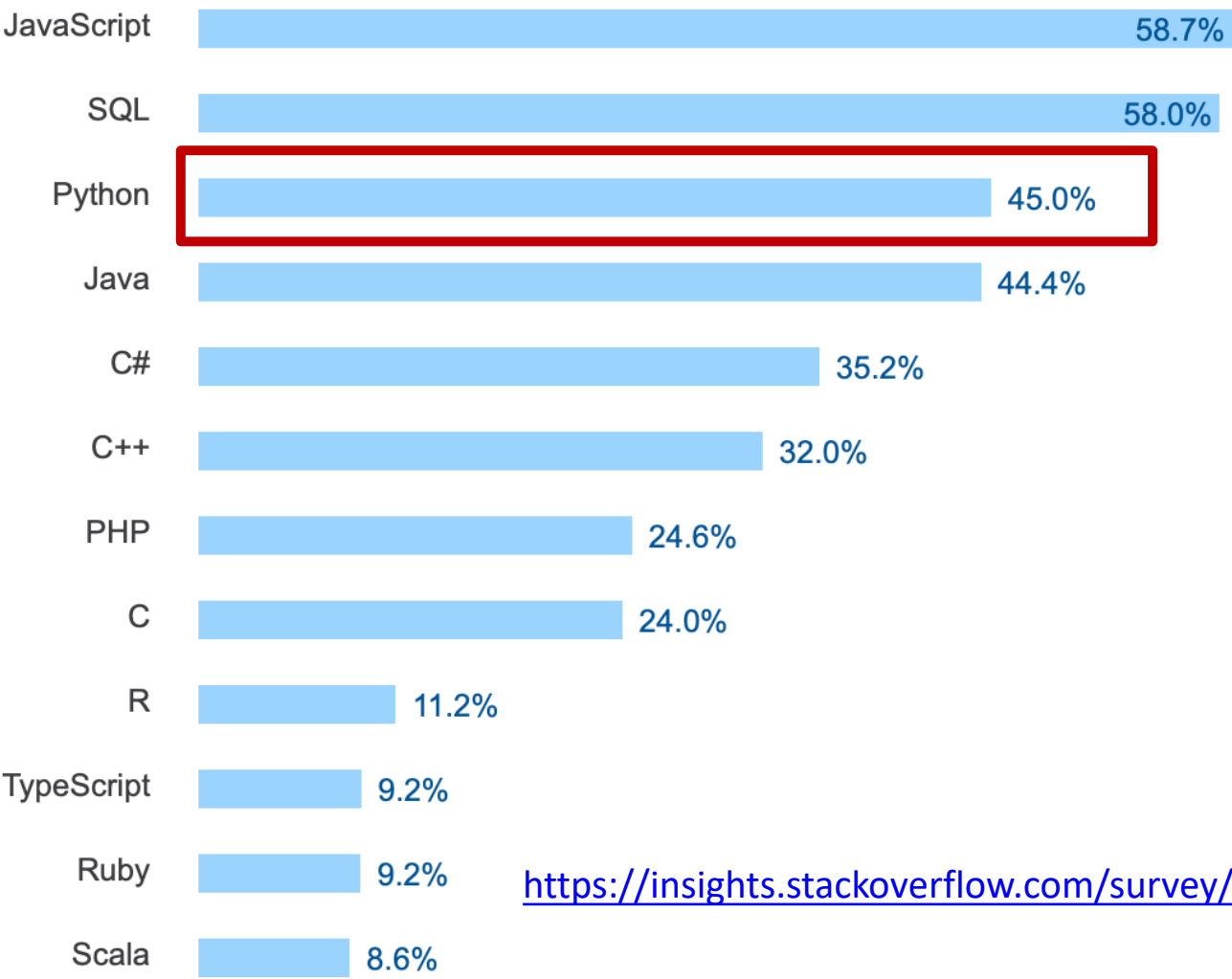
<https://docs.python.org/3/tutorial/modules.html>

python has applicability in

- Web and internet applications
- Scientific and numeric computing
- **Education**
- Graphical User Interface-based desktop applications
- Software development
- Business applications
- Operating systems
- Programming language development

<https://www.python.org/about/apps/>

Most Popular Languages by Occupation



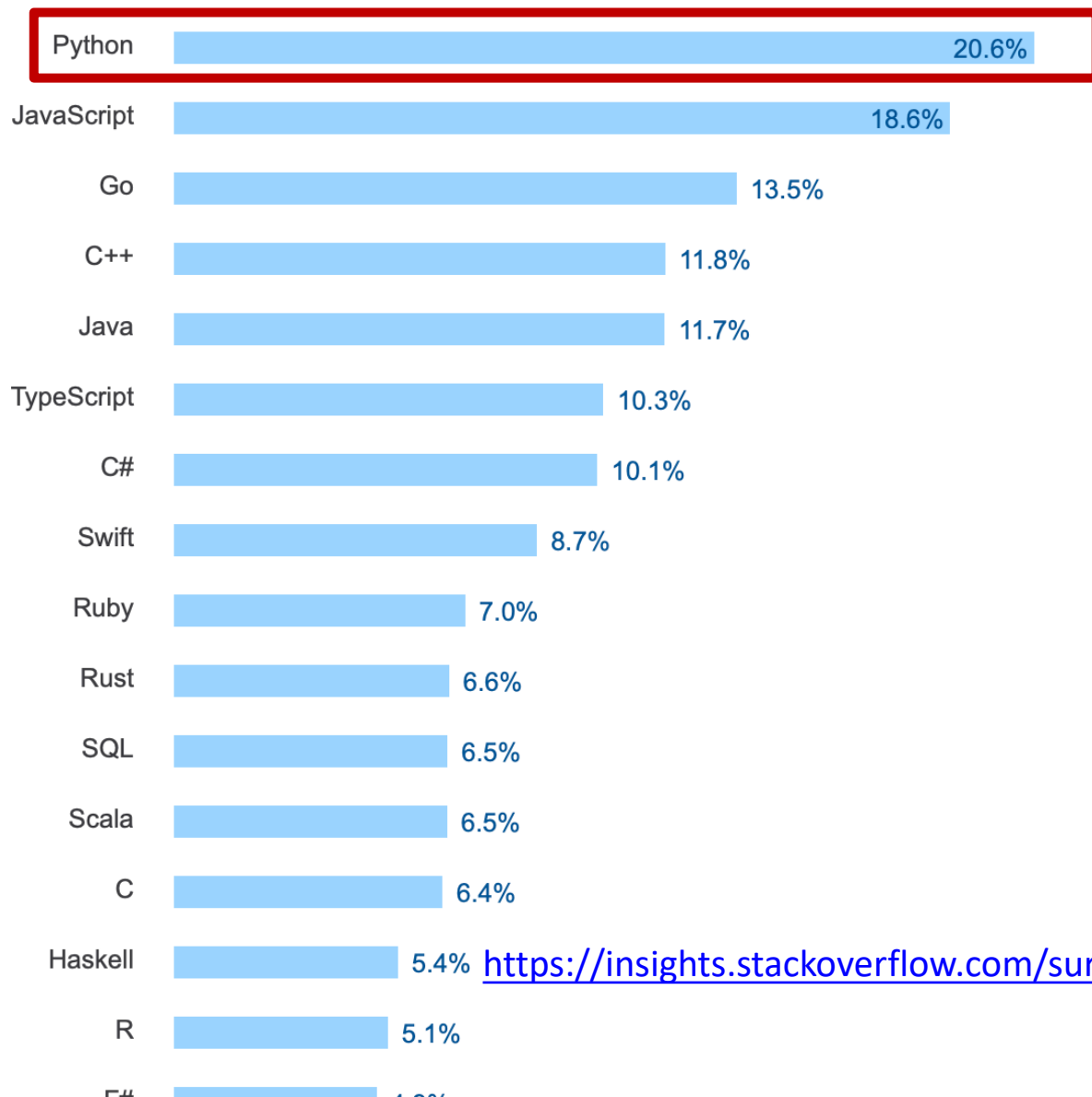
<https://insights.stackoverflow.com/survey/2017>

Most Loved, Dreaded, and Wanted Languages

Loved

Dreaded

Wanted



<https://insights.stackoverflow.com/survey/2017>

Who uses python?

- Google
- Quora
- Youtube
- Pinrest
- Instagram
- Amazon
- Facebook
- SurveyMonkeys

<https://www.quora.com/What-are-the-features-of-python>

<https://www.cleveroad.com/blog/discover-5-leading-companies-that-use-python-and-learn-does-it-fit-your-project>

Take home notions

the goal was to get familiar with the terminology

- Programming is fun (can be hard)
- Programing can be very useful
- It helps automate and speed up tasks
- Keys to problem solving:
 - Define well the problem (input, output)
 - Break down the problem in small manageable subproblems
- There is a lot of math and behind programming
- Flowcharts and generic tools that can help in algorithm development
- Pseudocode can help reduce implementation issues
 - 5 minutes of thinking can save hours of debugging time
 - when stuck, take a break and come back with a fresh perspective

Take home notions

- Many widely used programming languages are multi-paradigm
 - C++, Java, python, etc
 - they support object-oriented programming
 - typically in combination with imperative, procedural programming
- Python is a glue language that has key features that makes it suitable for data science
 - Is is generally awesome!!!
- More resources:
 - <http://planetpython.org/>
 - HOWTOS: <https://docs.python.org/3/howto/index.html>
 - Tutorial: <https://docs.python.org/3/tutorial/index.html>