Técnicas de Programação e Algoritmos

- Douglas Baptista de Godoy

/in/douglasbgodoy

github.com/douglasbgodoy

Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

Python Essentials 1: Module 1

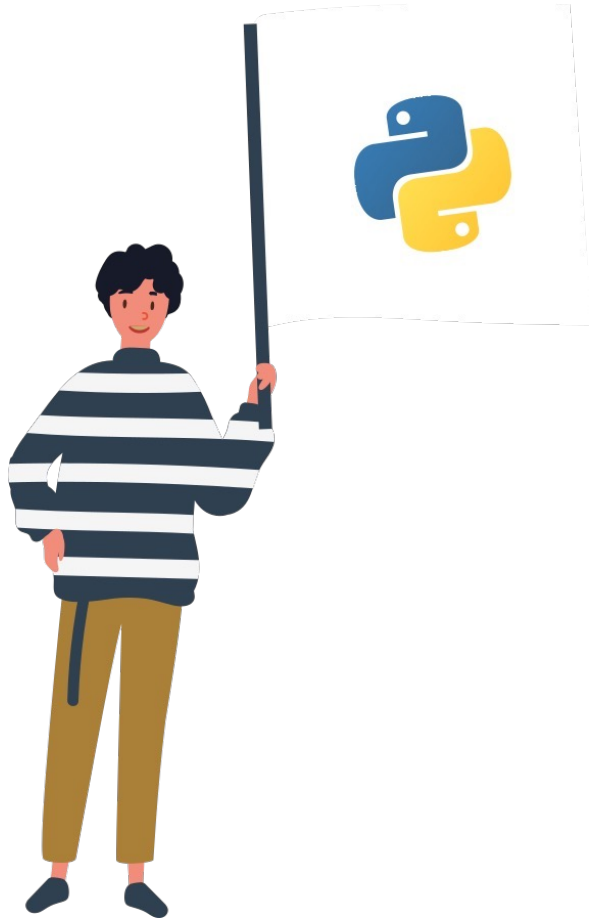
Introduction to Python and computer programming

Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

Index of Contents



Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

How does a computer program work?

A program makes a computer usable. Without a program, a computer, even the most powerful one, is nothing more than an object. Similarly, without a player, a piano is nothing more than a wooden box.

Computers are able to perform very complex tasks, but this ability is not innate.

It can execute only extremely simple operations. For example, a computer cannot understand the value of a complicated mathematical function by itself.

Language is the keyword

Imagine that you want to know the average speed you've reached during a long journey. You know the distance, you know the time, you need the speed.

Naturally, the computer will be able to compute this, but the computer is not aware of such things as distance, speed, or time. Therefore, it is necessary to instruct the computer to:

- Accept a number representing the distance.
- Accept a number representing the travel time.
- Divide the former value by the latter and store the result in the memory.
- Display the result.

These four simple actions form a program.

Natural languages vs. programming languages

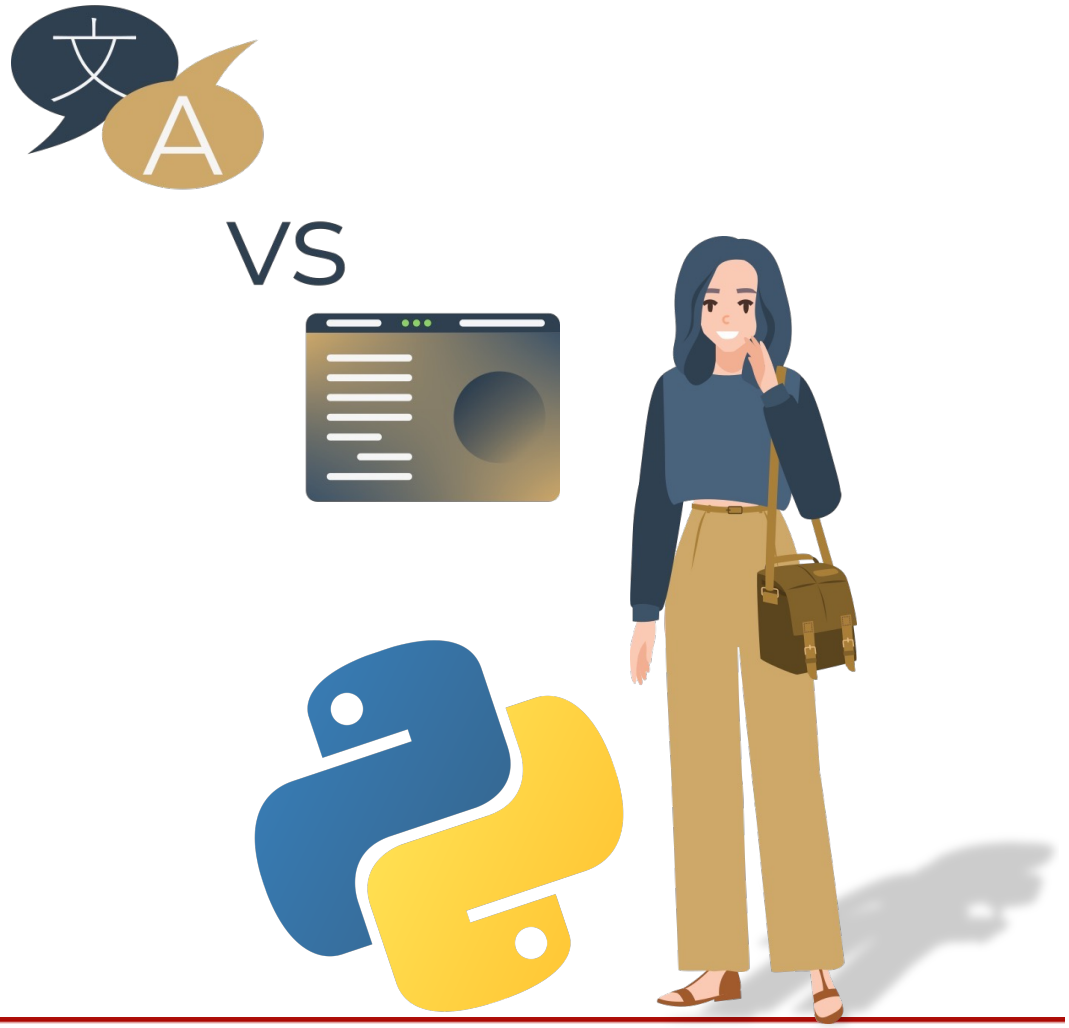
A language is a means (and a tool) for expressing and recording thoughts.

Computers have their own language, too, called machine language, which is very rudimentary.

The commands it recognizes are very simple. We can imagine that the computer responds to orders like "take that number, divide by another and save the result".

A complete set of known commands is called an instruction list, sometimes abbreviated to IL.

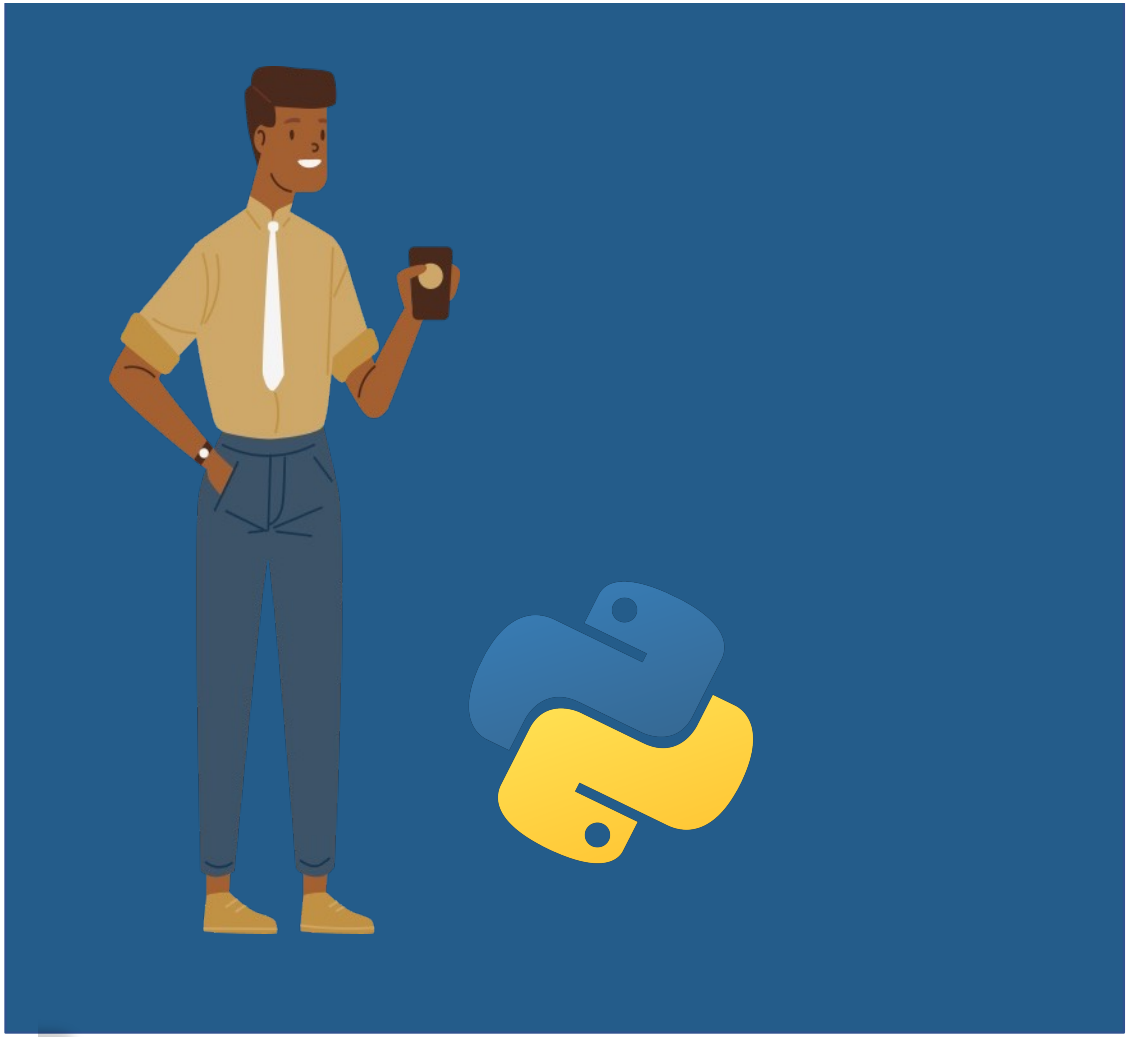
Note: machine languages are developed by humans.



Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo



What make a language?

We can say that each language consists of the following elements:

- An alphabet. A set of symbols used to build words.
- A lexis. A set of words the language offers its users
- A syntax. A set of rules.
- Semantics. A set of rules determining if a certain phrase makes sense.

The IL is, in fact, the alphabet of a machine language.

A program written in a high-level programming language is called a source code. Similarly, the file containing the source code is called the source file.

Compilation vs. Interpretation

Computer programming is the act of composing the selected programming language's elements in the order that will cause the desired effect.

Of course, such a composition has to be correct in many senses:

- Alphabetically.
- Lexically.
- Syntactically.
- Semantically.

Compilation vs. Interpretation

There are two different ways of transforming a program from a high-level programming language into machine language:

The source program is translated once by getting a file, containing the machine code; now you can distribute the file worldwide.

You can translate the source program each time it has to be run; the program performing this kind of transformation is called an interpreter, as it interprets the code every time it is intended to be executed.

What does the interpreter actually do?

- For example, if you try to use an entity of an unknown name, it will cause an error, but the error will be discovered in the place where it tries to use the entity, not where the new entity's name was introduced.

- If the line looks good, the interpreter tries to execute it (note: each line is usually executed separately, so the trio "read-check-execute" can be repeated many times
- - more times than the actual number of lines in the source file, as some parts of the code may be executed morethan once).

- It is also possible that a significant part of the code may be executed successfully before the interpreter finds an error. This is normal behavior in this execution model.

What does the interpreter actually do?

Let's assume once more that you have written a program. Now, it exists as a computer file: a computer program is actually a piece of text, so the source code is usually placed in text files.

The interpreter reads the source code in a way that is common in Western culture: from top to bottom and from left to right.

- First of all, the interpreter checks if all subsequent lines are correct (using the four aspects covered earlier).
- If the interpreter finds an error, it finishes its work

immediately. The only result in this case is an error message

The interpreter will inform you where the error is located and what caused it. However, these messages may be misleading, as the interpreter isn't able to follow your exact intentions, and may detect errors at some distance from their real causes.

Compilation vs. interpretation

Advantages and disadvantages

	Compilation	Interpretation
ADVANTAGES	The execution of the translated code is usually faster. Only the user has to have the compiler - the end-user may use the code without it. The translated code is stored using machine language.	You can run the code as soon as you complete it. The code is stored using programming language, not machine language.
DISADVANTAGES	The compilation itself may be a very time-consuming process. You have to have as many compilers as hardware platforms you want your code to be run on.	Don't expect interpretation to ramp up your code to high speed. Both you and the end user have to have the interpreter to run your code.

- Python is an interpreted language.
- if you want to program in Python, you'll need the Python interpreter.

What is Python?

- Python is a widely-used, interpreted, object-oriented, and high-level programming language with dynamic semantics, used for general- purpose programming.

- And while you may know the python as a large snake, the name of the Python programming language comes from an old BBC television comedy sketch series called Monty Python's Flying Circus.

- One of the amazing features of Python is the fact that it is actually one person's work.

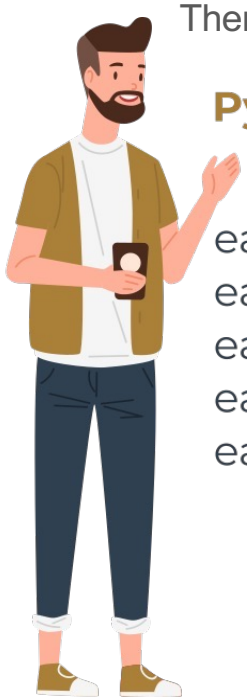
- Python was created by Guido van Rossum, born in 1956 in Haarlem, the Netherlands.

Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

What makes Python special?



There are many reasons:

Python is:

easy to learn,
easy to teach,
easy to use,
easy to understand,
easy to obtain.

Sobre a linguagem

- Criada por Guido van Rossum
- Origem do nome: grupo de humoristas *Monty Python*
- Sintaxe simples e fácil de ser assimilada
- Fácil de usar, aprender, ler
- Orientada à objetos, estruturada e funcional
- Interpretada Ambiente interativo

Mais sobre a linguagem

- Extremamente portável (Multiplataforma)
  - Unix/Linux, Windows, Mac.

  Licença GPL-compatível
- Tudo é objeto

  Pacotes, módulos, classes, funções

  Tratamento exceções

  Sobrecarga de operadores

  Identação para estrutura de bloco

    O resto é sintaxe convencional

Por que usar Python?

- Uma das linguagens mais divertidas que se tem atualmente
- Já vem com “baterias inclusas” (vasto repertório de bibliotecas)
- Protótipos rápidos sem preocupação com detalhes de implementação da linguagem
  - Linguagem Interpretada: evita “codifica-compila-roda”
- Bem menos linhas de código comparando com Java, C/C++...

Quem usa?

- Google (vários projetos)
- NASA (vários projetos)
- Muitas Universidades, como MIT, e Stanford
- Globo.com

Python rivals

Python has two direct competitors, with comparable properties and predispositions. These are:

- PEARL
- RUBY

Where can we see Python in action?

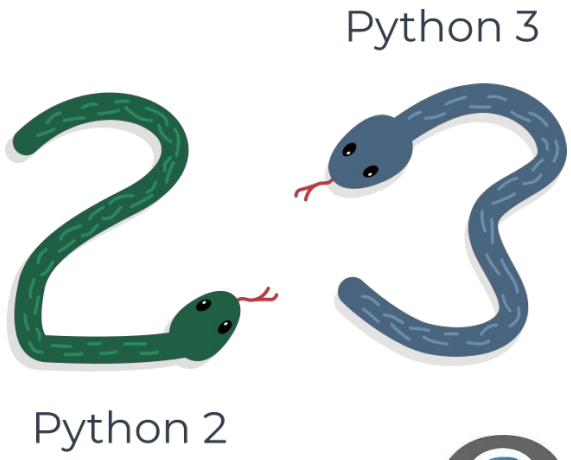
We see it every day and almost everywhere. It's used extensively to implement complex Internet services like search engines, cloud storage and tools, social media and so on. Whenever you use any of these services, you are actually very close to Python, although you wouldn't know it.

There is more than one Python.

There are two main kinds of Python, called Python 2 and Python 3.

Python 2 is an older version of the original Python. Its development has since been intentionally stalled, although that doesn't mean that there are no updates to it.

Python 3 is the newer version of the language. It's going through its own evolutionary path, creating its own standards and habits.



Jython.

"J" is for "Java".

PyPy and RPython.

Python within a Python.

Another Python family member is Cython.

Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

Python Essentials 1: Module 2

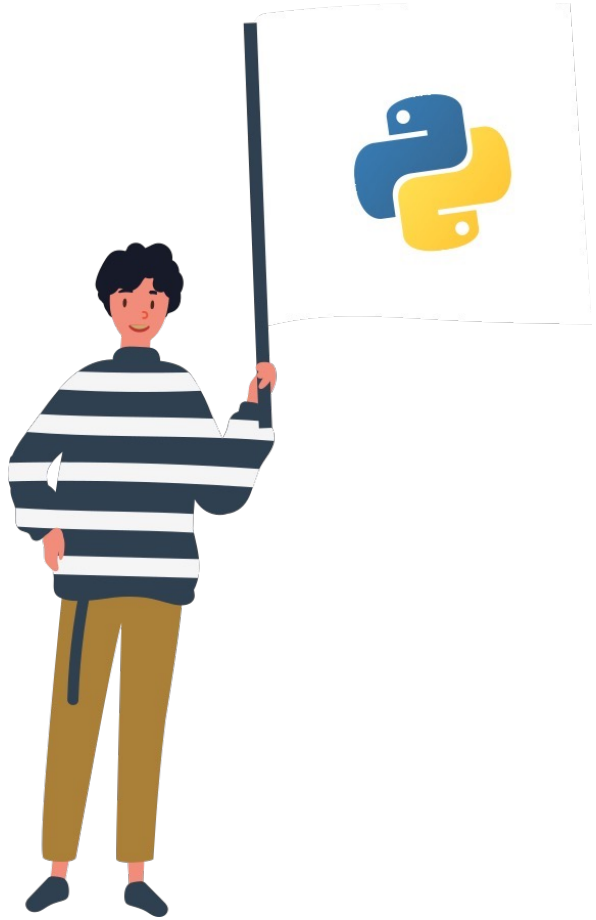
Data types, variables, Basic I/O operations, and basic operators

Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

Index of Contents



Your very first program

Comando de saída

It's time to start writing some real, working Python code. It'll be very simple for the time being.

Run the next code in the your editor window.

```
1  print("Hello, World!")
2
```

If everything goes okay here, you'll see this line of text in the console window.

```
Console >_

Hello, World!
```

Your very first program

As you can see, the first program consists of the following parts:

- The word print.
- An opening parenthesis.
- A quotation mark.
- A line of text: Hello, world!.
- Another quotation mark.
- A closing parenthesis.

Each of the above plays a very important role in the code.

The print() function

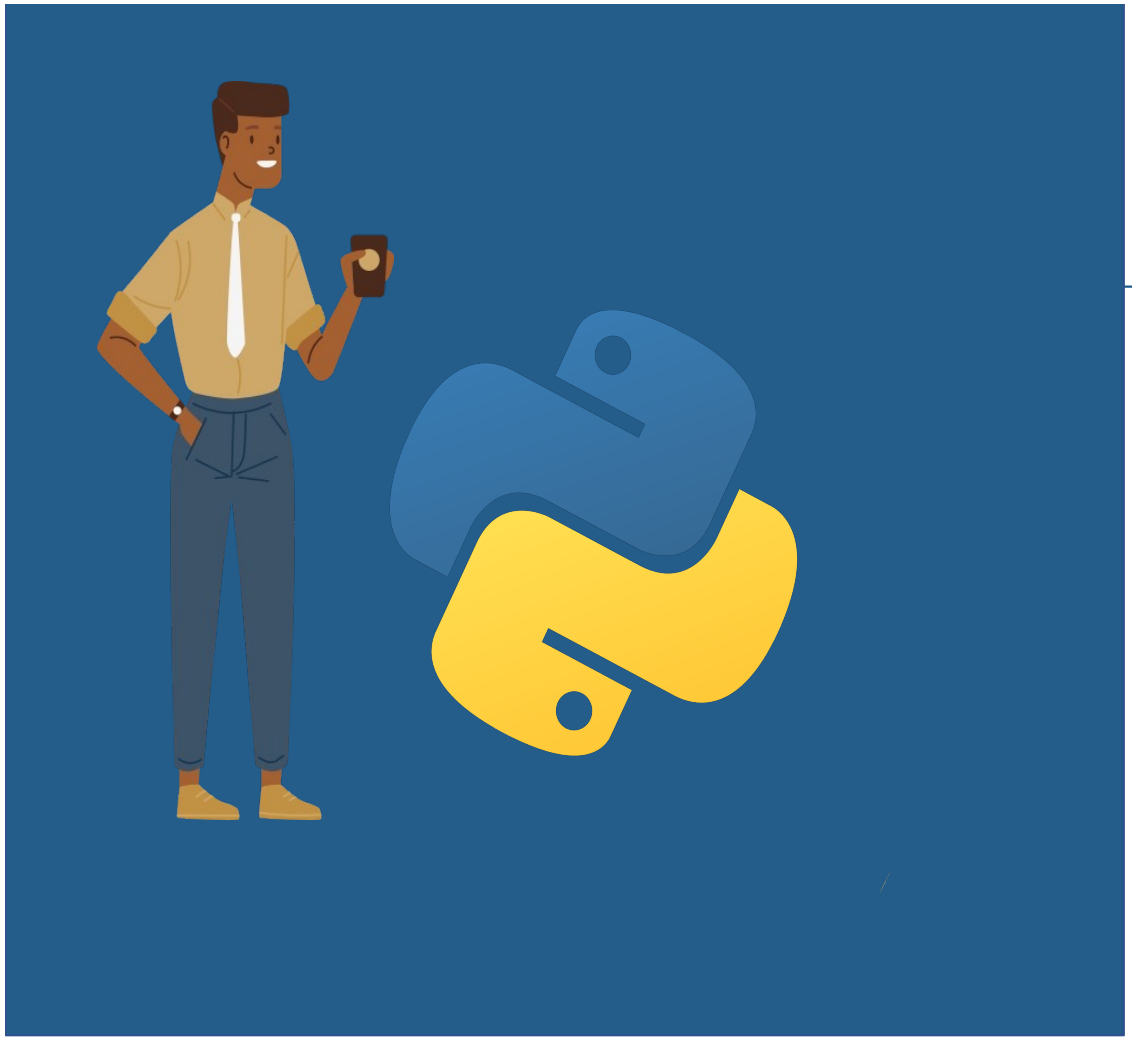
Look at the line of code below:

```
print("Hello, World!")
```

The word print that you can see here is a function name. That doesn't mean that wherever the word appears it is always a function name. The meaning of the word comes from the context in which the word has been used.

A function is a separate part of the computer code able to:

- Cause some effect.
- Evaluate a value and return it as the function's result.



Where do functions come from?

- They may come from Python itself; the print function is one of this kind.
- They may come from one or more of Python's add-ons, named modules.
- You can write them yourself, placing as many functions as you want and need inside your program to make it simpler.

The name of the function should be significant (the name of the print function is self-evident).

The print() function

As we said before, a function may have:

- An effect.
- A result.

There's also a third, very important, function component: the argument(s).

Python functions are more versatile. Depending on the individual needs, they may accept any number of arguments, as many as necessary to perform their tasks.

Note: to distinguish ordinary words from function names, place a pair of empty parentheses after their names, even if the corresponding function wants one or more arguments.

The print() function

What happens when Python encounters an invocation like this one below?

```
function_name(argument)
```

- First, Python checks if the name specified is legal.
- Second, Python checks if the function's requirements for the number of arguments allows you to invoke the function in this way.
- Third, Python leaves your code for a moment and jumps into the function you want to invoke; of course, it takes your argument(s) too and passes it/them to the function.
- Fourth, the function executes its code, causes the desired effect (if any), evaluates the desired result(s) (if any) and finishes its task.
- Finally, Python returns to your code (to the place just after the invocation) and resumes its execution.

The print() function

Three important questions have to be answered as soon as possible:

1. What is the effect the print() function causes?

- Takes its arguments.
- Converts them into human-readable form if needed.
- Sends the resulting data to the output device.

2. What arguments does print() expect?

Any. print() is able to operate with virtually all types of data offered by Python. Strings, numbers, characters, logical values, objects.

2. What arguments does print() expect?

None. Its effect is enough.

Examples

Sandbox

```
1  print("The itsy bitsy spider climbed up the waterspout.")
2  print("Down came the rain and washed the spider out.")
3
```

Console >_

```
The itsy bitsy spider climbed up the waterspout.
Down came the rain and washed the spider out.
```

Examples

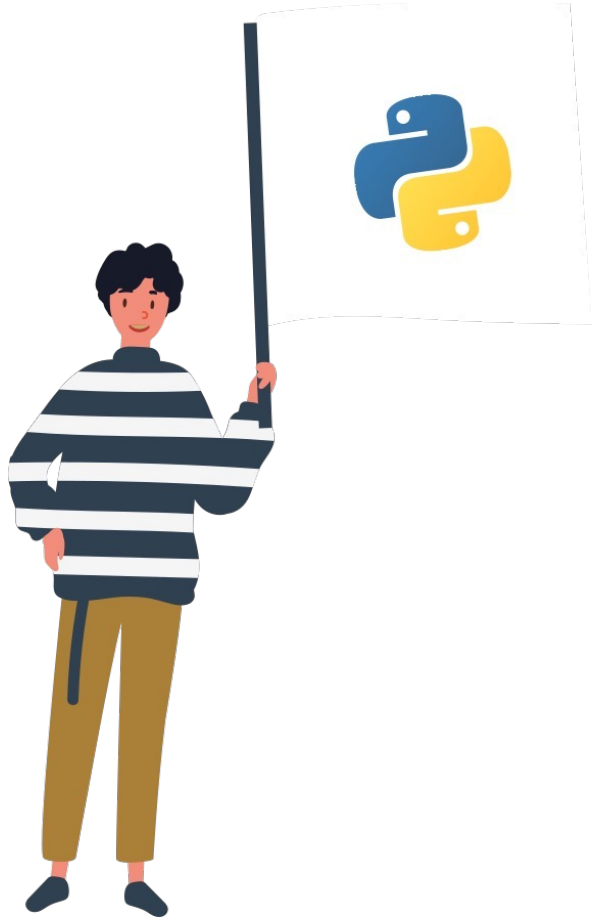
```
1  print("The itsy bitsy spider climbed up the waterspout.")
2  print()
3  print("Down came the rain and washed the spider out.")
4
```

Sandbox

```
The itsy bitsy spider climbed up the waterspout.


Down came the rain and washed the spider out.
```

Index of Contents



Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

The escape and newline characters (\n)

The backslash (\ ) and the n form a special symbol named a newline character, which urges the console to start a new output line.

```
1  print("The itsy bitsy spider\nclimbed up the waterspout.")
2  print()
3  print("Down came the rain\nand washed the spider out.")
4
```

```
The itsy bitsy spider
climbed up the waterspout.

Down came the rain
and washed the spider out.
```

Using multiple arguments

So far we have tested the print() function behavior with no arguments, and with one argument. It's also worth trying to feed the print() function with more than one argument.

```
print("The itsy bitsy spider" , "climbed up" , "the waterspout.")
```

There is one print() function invocation, but it contains three arguments. All of them are strings. The arguments are separated by commas.

In this case, the commas separating the arguments play a completely different role than the comma inside the string. The former is a part of Python's syntax, the latter is intended to be shown in the console.

```
The itsy bitsy spider climbed up the waterspout.
```

Keyword arguments

Keyword arguments are arguments whose meaning is not dictated by their location, but by a special word (keyword) used to identify them.

The end and sep parameters can be used for formatting the output of the print() function. The sep parameter specifies the separator between the outputted arguments (e.g., print("H", "E", "L", "L", "O", sep="-"), whereas the end parameter specifies what to print at the end of the print statement.

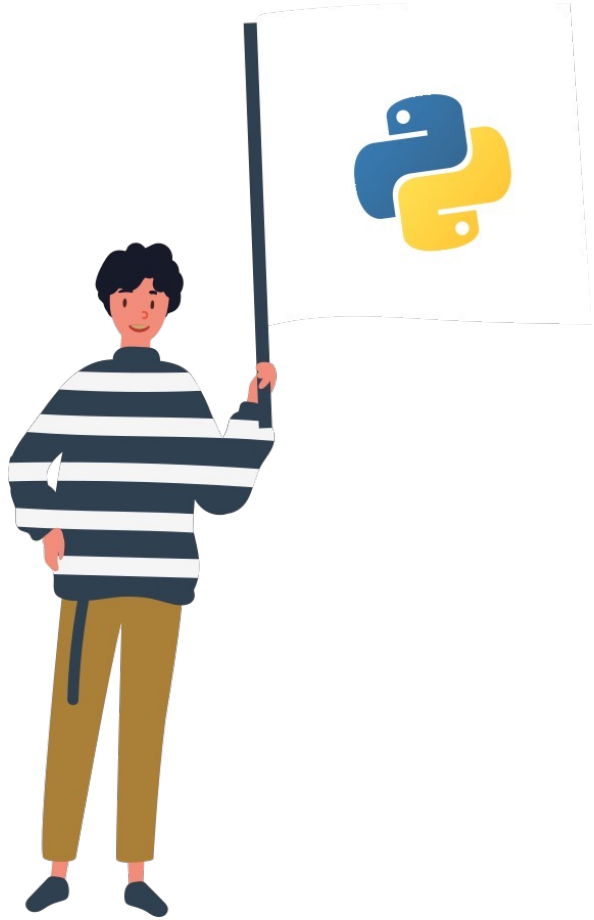
```
1 print("Meu", "nome", "é", "Monty", "Python.", sep="-")
2
```

```
Meu-nome-é-Monty-Python.
```

```
1 print("Meu nome é", "Python.", end="*")
2 print()
3 print("Monty Python.")
```

```
Meu nome é Python.*
Monty Python.
```

Index of Contents



Literals

- A literal is data whose values are determined by the literal itself.

- Literals are notations for representing some fixed values in code. Python has various types of literals - for example, a literal can be a number (numeric literals, e.g., 123), or a string (string literals, e.g., "I am a literal.").

```
1 print("2")
2 print(2)
```

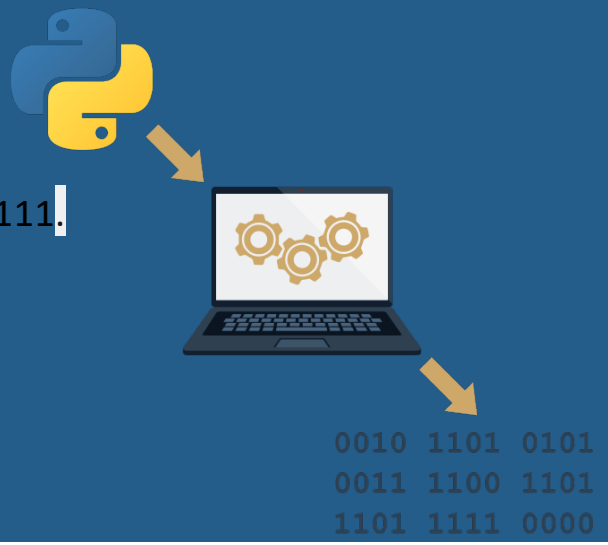
The print() function presents them in exactly the same way - this example is obvious, as their human-readable representation is also the same. Internally, in the computer's memory, these two values are stored in completely different ways - the string exists as just a string - a series of letters.

Binary, octal and hexadecimal numbers

We won't explore the intricacies of positional numeral systems here, but we'll say that the numbers handled by modern computers are of two types:

- **Integers**, that is, those which are devoid of the fractional part (e.g., 243, -1, 54, -87).

```
print(2)
print(20)
print(245)
```



número *onze milhões cento e onze mil cento e onze*
você pode escrever este número da seguinte forma: 11111111 ou desta forma: 11_111_111.

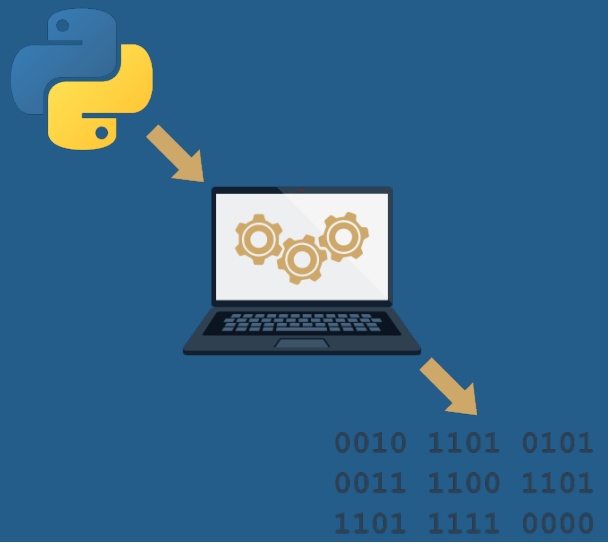
```
print(11_111_11)
```

Binary, octal and hexadecimal numbers

We won't explore the intricacies of positional numeral systems here, but we'll say that the numbers handled by modern computers are of two types:

- **Floating-point** numbers (or simply floats), that contain (or are able to contain) the fractional part (e.g., 27.5, -34.6).

```
print(2.5)
print(25.5)
print(45.5)
```



Binary, octal and hexadecimal numbers

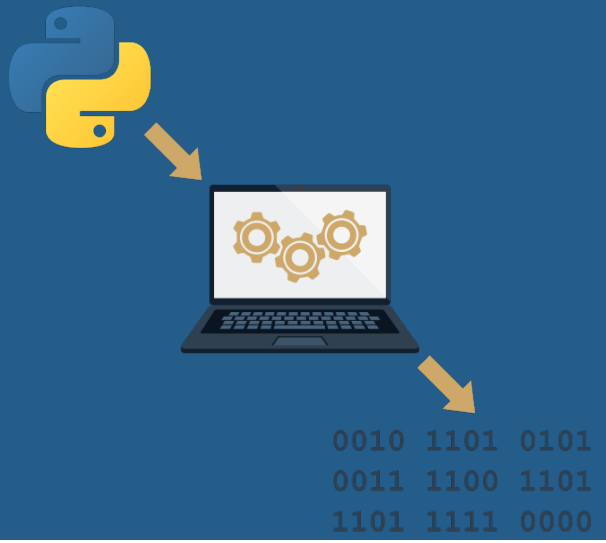
The **binary system** is a system of numbers that employs 2 as the base. Therefore, a binary number is made up of 0s and 1s only, (e.g., 1010 is 10 in decimal.).

O nome vem de George Boole (1815-1864), autor da obra fundamental, *The Laws of Thought*, que contém a definição de **álgebra booleana** – uma parte da álgebra que faz uso de apenas dois valores distintos: `True` e `False`, denotados como `1` e `0`.

```
1  print(True > False)
2  print(True < False)
3
4
```

Console >_

```
True
False
```



Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

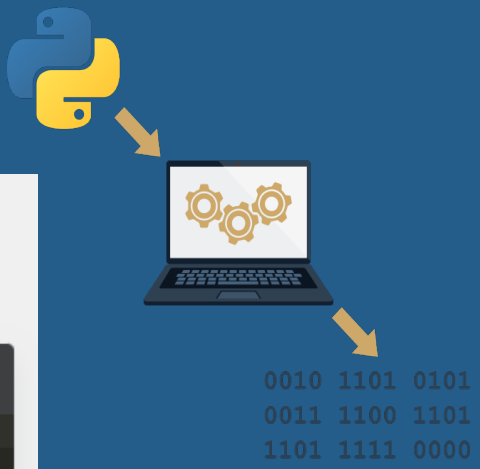
Binary, octal and hexadecimal numbers

Octal and hexadecimal numeration systems, similarly, employ 8 and 16 as their bases respectively. The hexadecimal system uses the decimal numbers and six extra letters.

Se um número inteiro for precedido por um prefixo `00` ou `0o` (zero-o), ele será tratado como um valor octal. Isso significa que o número deve conter dígitos retirados apenas do intervalo [0..7].

`0o123` é um número **octal** com um valor (decimal) igual a `83`.

```
1  print(0o123)
2
```

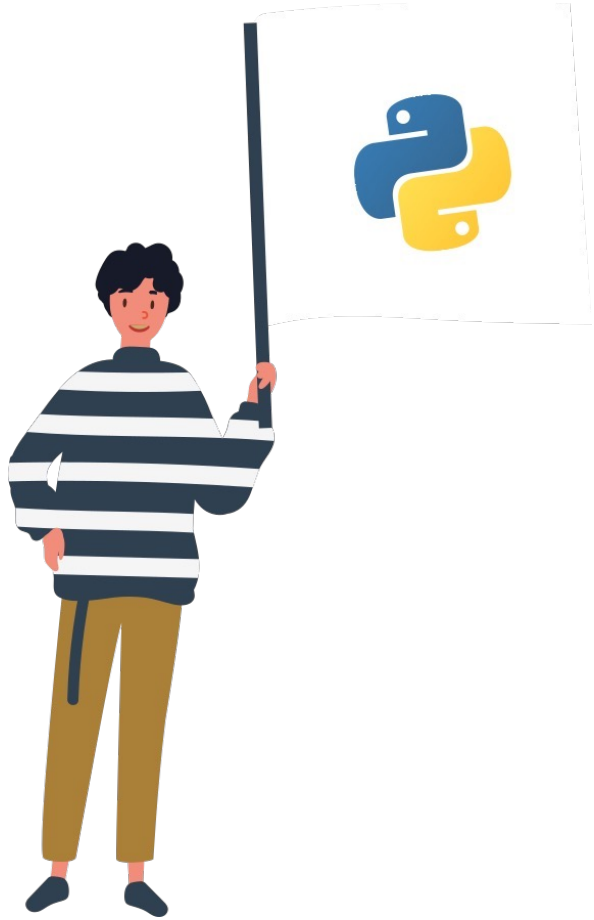


A segunda convenção nos permite usar números **hexadecimais**. Esses números devem ser precedidos pelo prefixo `0x` ou `0X` (zero-x).

`0x123` é um número **hexadecimal** com um valor (decimal) igual a `291`. A função `print()` também pode gerenciar esses valores. Tente isto:

```
1  print(0x123)
2
```

Index of Contents



Basic operators

An operator is a symbol of the programming language, which is able to operate on the values, creating its own standards and habits.

For example, just as in arithmetic, the + (plus) sign is the operator which is able to add two numbers, giving the result of the addition.

We'll begin with the operators which are associated with the most widely recognizable arithmetic operations:

`+` , `-` , `*` , `/` , `//` , `%` , `**`

An expression is a combination of values (or variables, operators, calls to functions – you will learn about them soon) which evaluates to a certain value. E.g., 1+2

Operators are special symbols or keywords which are able to operate on the values and perform (mathematical) operations, e.g., the * operator multiplies two values: x * y.

Basic operators

Arithmetic operators in Python:

- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `/` Classic division – always returns a float
- `%` Modulus
- `**` Exponentiation – left operand raised to the power of right operand
- `//` Floor or integer division – returns a number resulting from division

Remember: It's possible to formulate the following rules based on this result:

- When both ** arguments are integers, the result is an integer, too;
- When at least one ** argument is a float, the result is a float, too.

Examples

```
print(2 * 3)
print(2 * 3.)
print(2. * 3)
print(2. * 3.)
```

```
print(6 / 3)
print(6 / 3.)
print(6. / 3)
print(6. / 3.)
```

The result produced by the division operator is always a float.

Unary and binary operators

The subtraction operator is the - (minus) sign, although you should note that this operator also has another meaning - it can change the sign of a number.

A unary operator is an operator with only one operand, e.g., -1, or +3.

A binary operator is an operator with two operands, e.g., 4 + 5, or 12 % 5.

Operators and their priorities.

Most of Python's operators have left-sided binding, which means that the calculation of the expression is conducted from left to right.

```
print(9 % 6 % 2)
```

The result should be 1. This operator has left-sided binding. But there's one interesting exception.

Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

Exponentiation

Repeat the experiment, but now with exponentiation.

```
print(2 ** 2 ** 3)
```

The result shows that the exponentiation operator uses right-sided binding.

List of priorities

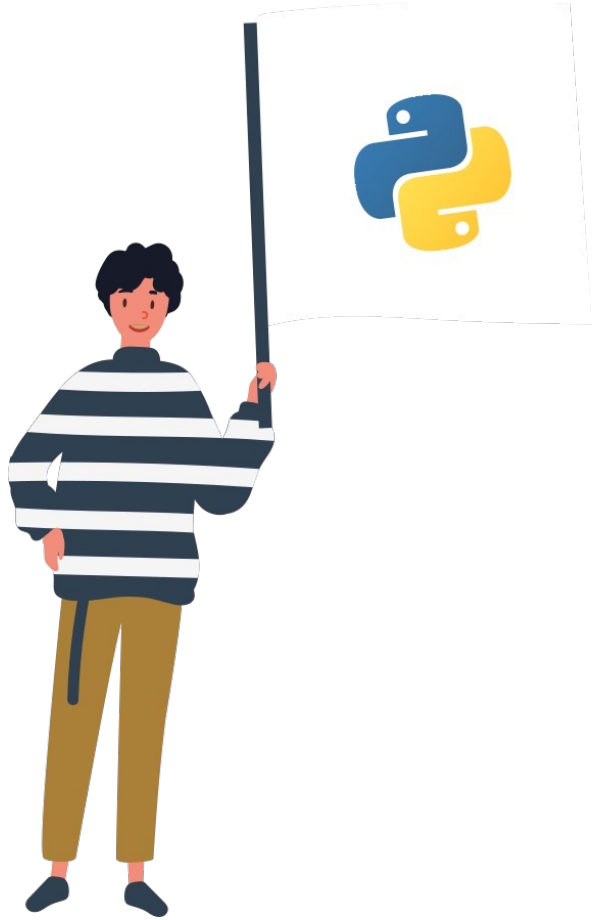
Priority	Operator	
1	`**`	
2	`+`, `-` (note: unary operators located next to the right of the power operator bind more strongly)	unary
3	`*`, `/`, `//`, `%`	
4	`+`, `-`	binary

- The ** operator (exponentiation) has the highest priority
- Then the unary + and -
- Then *, /, //, and %
- And, finally, the lowest priority: the binary + and -

Subexpressions in parentheses are always calculated first.

The exponentiation operator uses right-sided binding.

Index of Contents



Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

What are variables?

- A variable is a named location reserved to store values in the memory. A variable is created or initialized automatically when you assign a value to it for the first time.

- If you want to give a name to a variable, you must follow some strict rules:

- The name of the variable must be composed of upper-case or lower-case letters, digits, and the character _ .
- The name of the variable must begin with a letter.
- The underscore character is a letter.
- Upper- and lower-case letters are treated as different.
- The name of the variable must not be any of Python's reserved words.

Each variable must have a unique name - an identifier. A legal identifier must be a non-empty sequence of characters, must begin with an underscore(_) or a letter, and it cannot be a Python keyword. Identifiers in Python are case-sensitive.

Take a look at the list of words that play a very special role in every Python program.

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

What are variables?

Python is a dynamically-typed language, which means you don't need to declare variables in it. To assign values to variables, you can use a simple assignment operator in the form of the equal (=) sign, i.e., var = 1.

You can assign new values to already existing variables using the assignment operator or one of the compound operators, e.g.:

```
var = 2
print(var)

var = 3
print(var)

var += 1
print(var)
```

You can combine text and variables using the + operator, and use the print() function to output strings and variables, e.g.:

```
var = "007"
print("Agent " + var)
```

What are variables?

Aqui estão alguns nomes de variáveis **corretos**, mas nem sempre convenientes:

- `MyVariable`
- `i`
- `l`
- `t34`
- `Exchange_Rate`
- `counter`
- `days_to_christmas`
- `TheNameIsTooLongAndHardlyReadable`
- `_`

Esses nomes de variáveis também estão **corretos**:

- `Adiós_Señora`
- `sûr_la_mer`
- `Einbahnstraße`
- `переменная`.

O Python permite que você use não apenas letras latinas, mas também caracteres específicos de idiomas que usam outros alfabetos.

E agora, alguns nomes **incorretos**:

- `10t` (não começa com uma letra)
- `!important` (não começa com uma letra)
- `exchange rate` (contém um espaço).

Função type()

A função type() em Python retorna o tipo de um objeto.

```
#Exemplo 6
idade = 20
print(type(idade))

#Exemplo 7
idade = "20"
print(type(idade))

#Exemplo 8
altura = 1.87
print(type(altura))

#Exemplo 9
altura = float("1.87")
print(type(altura))
```

```
#Exemplo 10
idade = int(20.8)
print(type(idade))

#Exemplo 11
texto = str(20)
print(type(texto))

#Exemplo 12
teste = True
print(type(teste))
```

What are variables?

Você também pode usar operadores de atribuição compostos (operadores de atalho) para modificar valores atribuídos a variáveis.

You can also use **compound assignment operators** (shortcut operators) to modify values assigned to variables, e.g., var += 1, or var /= 5 * 2.

```
i = i + 2 * j  ⇒  i += 2 * j

var = var / 2  ⇒  var /= 2

rem = rem % 10  ⇒  rem %= 10

j = j - (i + var + rem)  ⇒  j -= (i + var + rem)

x = x ** 2  ⇒  x **= 2
```

Operadores Aritméticos

Operadores de atribuição compostos (operadores de atalho)

Operador	Exemplo	Comentário
+ =	x + = y	Equivale a X = X + Y.
- =	x - = y	Equivale a X = X – Y.
* =	x * = y	Equivale a X = X * Y.
/ =	x / = y	Equivale a X = X / Y.
% =	x % = y	Equivale a X = X % Y.

Comentários em Python

- Comentários são textos que podem ser inseridos em programas com o objetivo de documentá-los. Eles não são analisados pelo interpretador.
- Os comentários podem ocupar uma ou várias linhas, devendo ser inseridos nos programas utilizando-se os símbolos """ ..... """ ou #.
- **Exemplo:** # comentário de uma linha

  """ comentário de múltiplas linhas """

```
File  Edit  Format  Run  Options  Window  Help
print("Olá Python")
"""print("Comentários em Python")
print("Comentários 01")
print("Comentários de múltiplas linhas ")
print("Fim do Programa")"""
```

Leaving comments in code

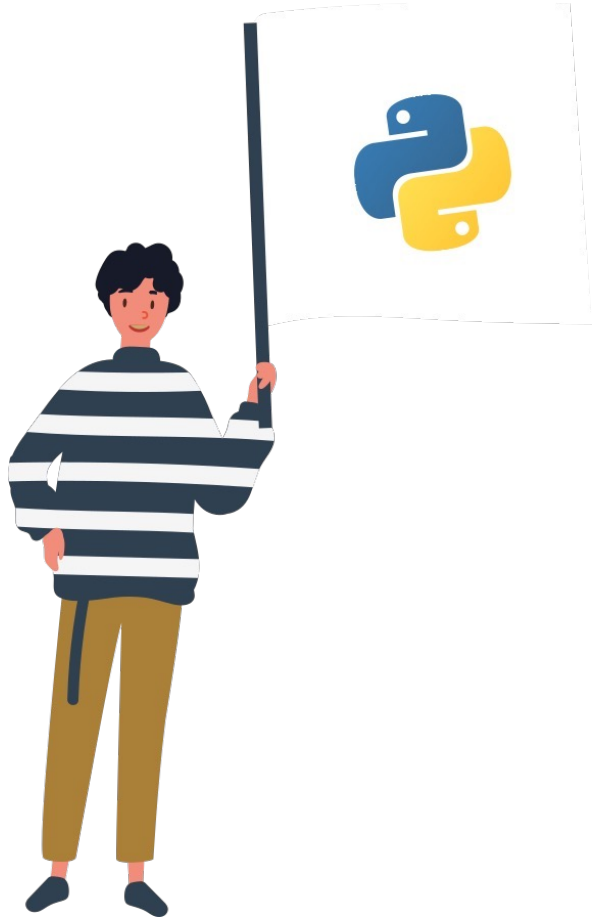
You may want to put in a few words addressed not to Python but to humans, usually to explain to other readers of the code how the tricks used in the code work, or the meanings of the variables.

In Python, a comment is a piece of text that begins with #. The comment extends to the end of line.

```
# This program prints
# an introduction to the screen.
print("Hello!")  # Invoking the print() function
# print("I'm Python.")
```

It's important to use comments to make programs easier to understand, and to use readable and meaningful variable names in code. However, it's equally important not to use variable names that are confusing, or leave comments that contain wrong or incorrect information!

Index of Contents



Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

The input() function

Comando de entrada

The input() function is able to read data entered by the user and to return the same data to the running program.

The program can manipulate the data, making the code truly interactive. Take a look at our example:

```
print("Tell me anything...")
anything = input()
print("Hmm...", anything, "... Really?")
```

The program prompts the user to input some data from the console.

The input() function is invoked without arguments; the function will switch the console to input mode; you'll see a blinking cursor, and you'll be able to input some keystrokes, finishing off by hitting the Enter key; all the inputted data will be sent to your program through the function's result;

Note: you need to assign the result to a variable; this is crucial - missing out this step will cause the entered data to be lost.

The input() function with an argument

The input() function can do something else: it can prompt the user without any help from print().

```
anything = input("Tell me anything...")
print("Hmm...", anything, "...Really?")
```

- The input() function is invoked with one argument - it's a string containing a message;
- The message will be displayed on the console before the user is given an opportunity to enter anything;
- input() will then do its job;

Type casting

Python offers two simple functions to specify a type of data and solve this problem - here they are: int() and float().

The int() function takes one argument (e.g., a string: int(string)) and tries to convert it into an integer. If it fails, the whole program will fail too.

The float() function takes one argument (e.g., a string: float(string)) and tries to convert it into a float (the rest is the same).

```
1 anything = float(input("Enter a number: "))
2 something = anything ** 2.0
3 print(anything, "to the power of 2 is", something)
4
```

Concatenation

The + (plus) sign, when applied to two strings, becomes a concatenation operator:

It simply concatenates (glues) two strings into one. Of course, like its arithmetic sibling, it can be used more than once in one expression, and in such a context it behaves according to left- sided binding.

```
fnam = input("May I have your first name, please? ")
lnam = input("May I have your last name, please? ")
print("Thank you.")
print("\nYour name is " + fnam + " " + lnam + ".")
```

Note: using + to concatenate strings lets you construct the output in a more precise way than with a pure print() function, even if enriched with the end= and sep= keyword arguments.

Replication

The result of the input() function is a string. You can add strings to each other using the concatenation (+) operator.

Check out this code:

```
num_1 = input("Enter the first number: ") # Enter 12
num_2 = input("Enter the second number: ") # Enter 21

print(num_1 + num_2) # the program returns 1221
```

You can also multiply (* - replication) strings, e.g.:

```
my_input = input("Enter something: ") # Example input: hello
print(my_input * 3) # Expected output: hellohellohello
```

What is the output of the following snippet?

```
x = int(input("Enter a number: ")) # The user enters 2
print(x * "5")
```

ANSWER = 55

Referências Bibliográficas

- MENEZES, Nilo Ney Coutinho, **Introdução à Programação com Python**, Novatec Editora, 4ª edição.

- CISCO NETWORKING ACADEMY - Python Essentials 1, Disponível em: https://skillsforall.com/pt/course/python-essentials-1?courseLang=en-US. Acesso em: 21 fevereiro.2024.

- www.pythoninstitute.org

PYTHON INSTITUTE
OPEN EDUCATION & DEVELOPMENT GROUP

Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo