1.4 Programming Language Characteristics

2016-0606 INFO W18: Python Bridge

Different Programming Languages

Different Programming Languages

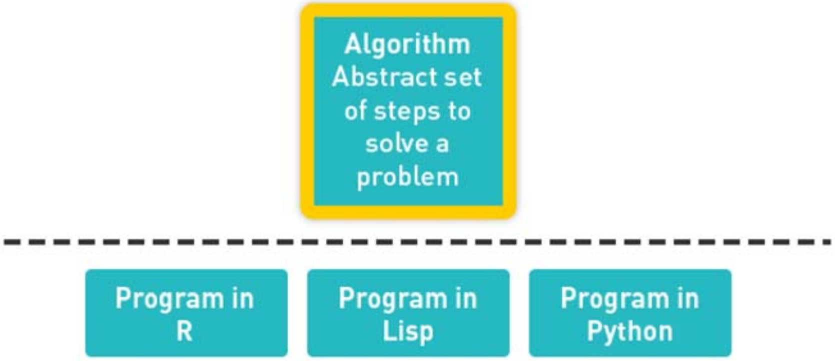
- The algorithm is abstract—it's the set of steps that's separate from how we communicate it.

Different Programming Languages

- The algorithm is abstract—it's the set of steps that's separate from how we communicate it.
    - The same algorithm could be encoded in many programming languages.

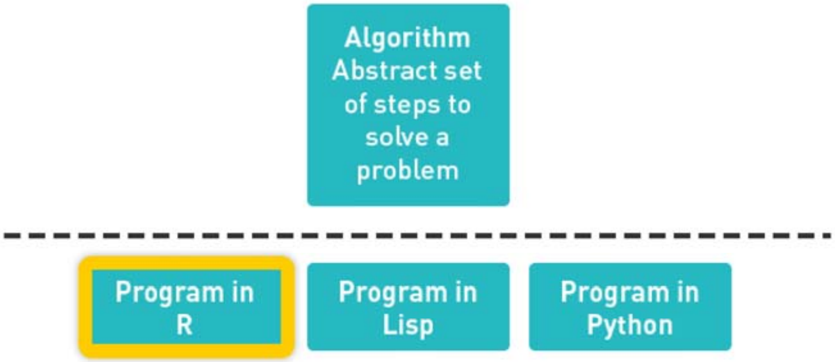
Different Programming Languages

- The algorithm is abstract—it's the set of steps that's separate from how we communicate it.
    - The same algorithm could be encoded in many programming languages.



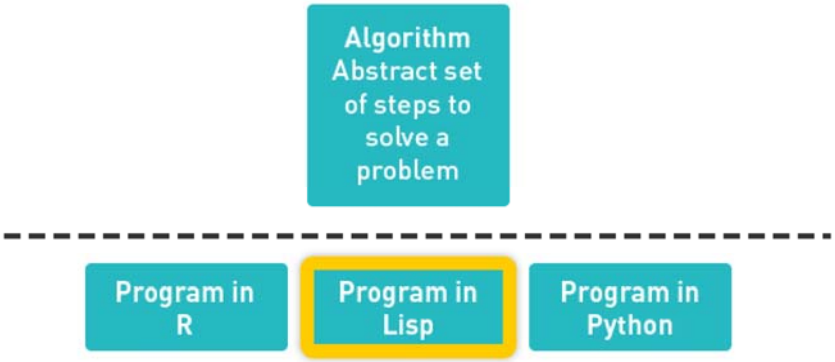
Different Programming Languages

- The algorithm is abstract—it's the set of steps that's separate from how we communicate it.
    - The same algorithm could be encoded in many programming languages.



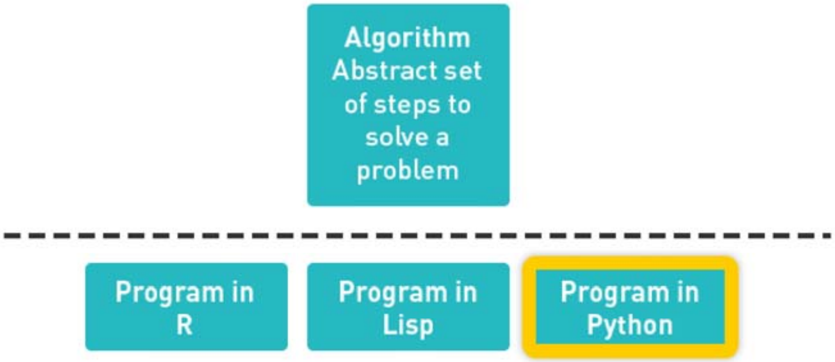
Different Programming Languages

- The algorithm is abstract—it's the set of steps that's separate from how we communicate it.
  - The same algorithm could be encoded in many programming languages.



Different Programming Languages

- The algorithm is abstract—it's the set of steps that's separate from how we communicate it.
  - The same algorithm could be encoded in many programming languages.



Different Programming Languages (cont.)

- Each programming language has a different syntax and is good at different things.

Different Programming Languages (cont.)

- Each programming language has a different syntax and is good at different things.

- How do you choose a language?

Different Programming Languages (cont.)

- Each programming language has a different syntax and is good at different things.
- How do you choose a language?
  - One concern is whether you want a low-level or a high-level language.

Low Level vs. High Level

Low Level vs. High Level

- Low-level languages give you control over the details of a computer.

Low Level vs. High Level

- Low-level languages give you control over the details of a computer.
  - E.g., memory allocation, memory addresses, processor sharing, computer hardware, etc.

Low Level vs. High Level

- Low-level languages give you control over the details of a computer.
  - E.g., memory allocation, memory addresses, processor sharing, computer hardware, etc.
- These languages run fast.

Low Level vs. High Level

- Low-level languages give you control over the details of a computer.
  - E.g., memory allocation, memory addresses, processor sharing, computer hardware, etc.
- These languages run fast.

- It also means the programmer has to know things about the hardware the code is running on.

Low Level vs. High Level

- Low-level languages give you control over the details of a computer.
  - E.g., memory allocation, memory addresses, processor sharing, computer hardware, etc.
- These languages run fast.
- It also means the programmer has to know things about the hardware the code is running on.

- The programmer also has more details to worry about.

Low Level vs. High Level

- Low-level languages give you control over the details of a computer.
  - E.g., memory allocation, memory addresses, processor sharing, computer hardware, etc.
- These languages run fast.
- It also means the programmer has to know things about the hardware the code is running on.
- The programmer also has more details to worry about.
  - There is more code to write.

Low Level vs. High Level

- Low-level languages give you control over the details of a computer.
  - E.g., memory allocation, memory addresses, processor sharing, computer hardware, etc.
- These languages run fast.
- It also means the programmer has to know things about the hardware the code is running on.
- The programmer also has more details to worry about.
  - There is more code to write.
  - There are a lot of places to make mistakes.

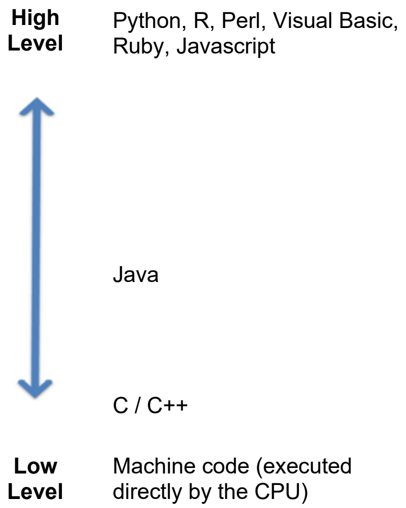
Low Level vs. High Level

- Low-level languages give you control over the details of a computer.
  - E.g., memory allocation, memory addresses, processor sharing, computer hardware, etc.
- These languages run fast.
- It also means the programmer has to know things about the hardware the code is running on.
- The programmer also has more details to worry about.
  - There is more code to write.
  - There are a lot of places to make mistakes.

- Higher level languages abstract away from these details, making many tasks automatic.

Low Level vs. High Level (cont.)

Low Level vs. High Level (cont.)



High Level

Python, R, Perl, Visual Basic, Ruby, Javascript

Java

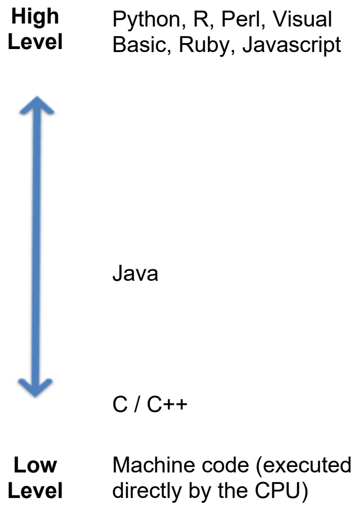
C / C++

Low Level

Machine code (executed directly by the CPU)

Low Level vs. High Level (cont.)

- The higher up you go in abstraction, the more choices you have for a language.



High Level

Python, R, Perl, Visual Basic, Ruby, Javascript

Java

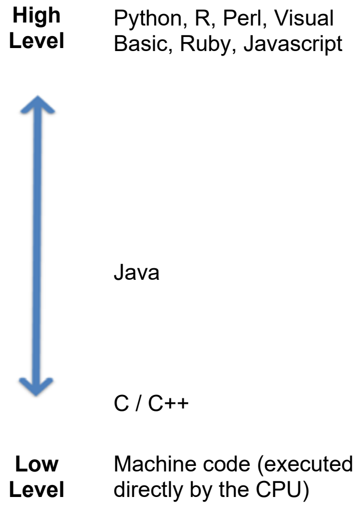
C / C++

Low Level

Machine code (executed directly by the CPU)

Low Level vs. High Level (cont.)

- The higher up you go in abstraction, the more choices you have for a language.
- At the bottom, there's machine language—the code that's executed by a computer CPU.



High Level

Python, R, Perl, Visual Basic, Ruby, Javascript

Java

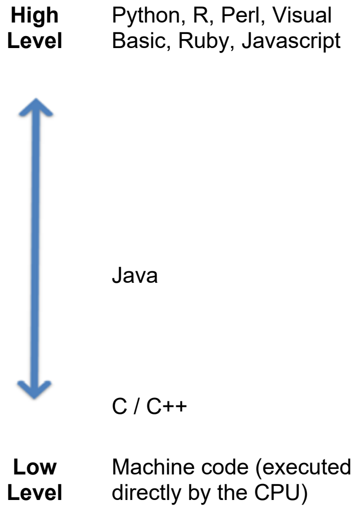
C / C++

Low Level

Machine code (executed directly by the CPU)

Low Level vs. High Level (cont.)

- The higher up you go in abstraction, the more choices you have for a language.
- At the bottom, there's machine language—the code that's executed by a computer CPU.



High Level Python, R, Perl, Visual Basic, Ruby, Javascript

Java

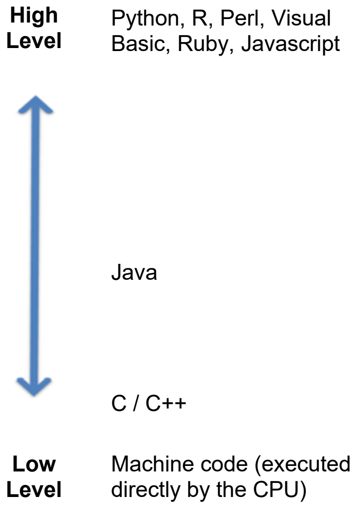
C / C++

Low Level Machine code (executed directly by the CPU)

Low Level vs. High Level (cont.)

- The higher up you go in abstraction, the more choices you have for a language.
- At the bottom, there's machine language—the code that's executed by a computer CPU.

- A lot of performance-oriented work takes place in C++.



High Level Python, R, Perl, Visual Basic, Ruby, Javascript

Java

C / C++

Low Level Machine code (executed directly by the CPU)

Low Level vs. High Level (cont.)

- The higher up you go in abstraction, the more choices you have for a language.
- At the bottom, there's machine language—the code that's executed by a computer CPU.
- A lot of performance-oriented work takes place in C++.

- Above this is Java, which is similar, but a bit higher level.

High Level Python, R, Perl, Visual Basic, Ruby, Javascript

Java

C / C++

Low Level Machine code (executed directly by the CPU)

Low Level vs. High Level (cont.)

- The higher up you go in abstraction, the more choices you have for a language.
- At the bottom, there's machine language—the code that's executed by a computer CPU.
- A lot of performance-oriented work takes place in C++.
- Above this is Java, which is similar, but a bit higher level.

- Then there is a proliferation of higher level languages, including Python.

High Level Python, R, Perl, Visual Basic, Ruby, Javascript

Java

C / C++

Low Level Machine code (executed directly by the CPU)

Machine Code

Machine Code

- Machine code is the lowest level language.

Machine Code

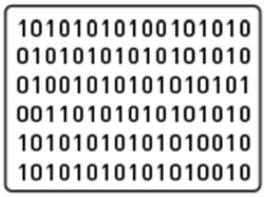
- Machine code is the lowest level language.
- It's as unreadable to humans as code gets—just sequences of bits.

Machine Code

- Machine code is the lowest level language.
- It's as unreadable to humans as code gets—just sequences of bits.

Machine Code

- Machine code is the lowest level language.
- It's as unreadable to humans as code gets—just sequences of bits.

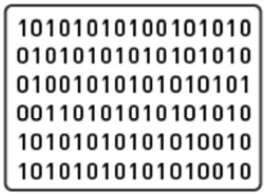


Machine Language

- Usually, a programmer would program in a higher level language, like assembly language, that's then compiled into machine language.

Machine Code

- Machine code is the lowest level language.
- It's as unreadable to humans as code gets—just sequences of bits.



Machine Language

- Usually, a programmer would program in a higher level language, like assembly language, that's then compiled into machine language.
- This is a language that only very specialized programmers ever deal with.

High-Level Languages

High-Level Languages

- Higher level languages, like Python, C, and Java, look more like human language and are easier to understand.

High-Level Languages

- Higher level languages, like Python, C, and Java, look more like human language and are easier to understand.
- They automate low-level tasks for the programmer.

High-Level Languages

- Higher level languages, like Python, C, and Java, look more like human language, and are easier to understand.
- They automate low-level tasks for the programmer.
  - For example, memory is allocated automatically.

High-Level Languages

- Higher level languages, like Python, C, and Java, look more like human language, and are easier to understand.
- They automate low-level tasks for the programmer.
  - For example, memory is allocated automatically.
    - You don't need to ever know what memory address you're using.

High-Level Languages

- Higher level languages, like Python, C, and Java, look more like human language, and are easier to understand.
- They automate low-level tasks for the programmer.
  - For example, memory is allocated automatically.
    - You don't need to ever know what memory address you're using.
    - When it's not needed any more, it's automatically "garbage collected" so it can be used again.

High-Level Languages (cont.)

- When the programmer opens a data file in Python using `open(filename)`, the computer will begin moving part of the file from a disk to a memory buffer, so that it can be accessed faster when the programmer wants to start reading.

42 of 86

Abstracting From Hardware

43 of 86

Abstracting From Hardware

- High-level languages also abstract away from the computer hardware.

44 of 86

Abstracting From Hardware

- High-level languages also abstract away from the computer hardware.
    - You don't need to worry about different types of memory, different processors, where components are located, etc.

45 of 86

Abstracting From Hardware

- High-level languages also abstract away from the computer hardware.
    - You don't need to worry about different types of memory, different processors, where components are located, etc.
- This is important because computer hardware is always changing, and your code may have to run on many different types of hardware.

Defining Computers

Defining Computers

- Many programming classes begin by defining what a computer is.

Defining Computers

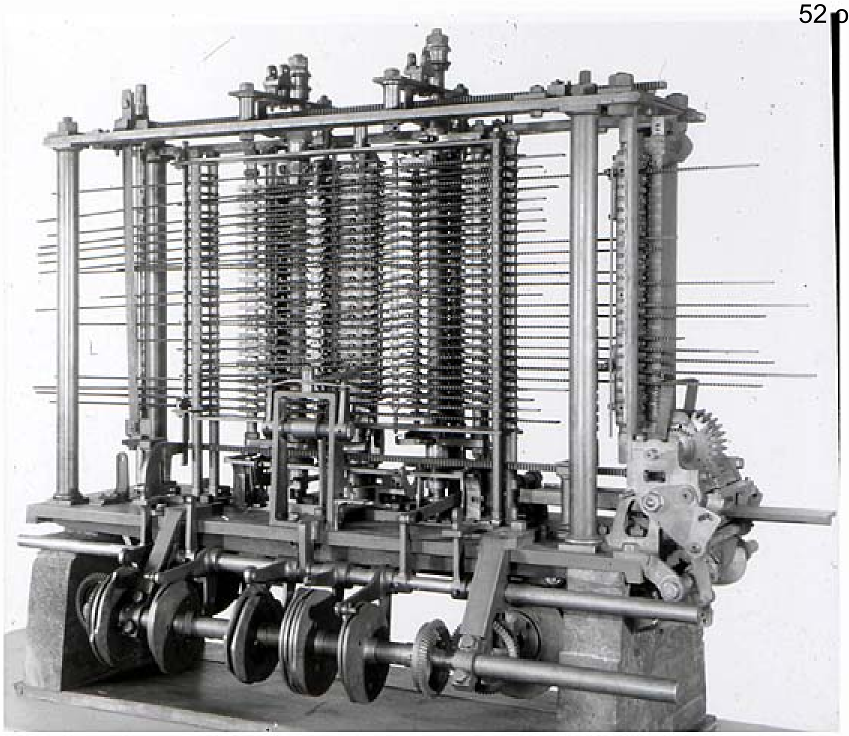
- Many programming classes begin by defining what a computer is.
- This isn't easy. A modern computer usually has the same components: a central processing unit, memory, input/output, but these vary.

Defining Computers

- Many programming classes begin by defining what a computer is.
- This isn't easy. A modern computer usually has the same components: a central processing unit, memory, input/output, but these vary.
- At one time, computations were carried out by humans in large warehouses—and they were also called computers.

Defining Computers (cont.)

- Later, mechanical computers were designed, such as the difference engine, pioneered by Charles Babbage (1791–1871).



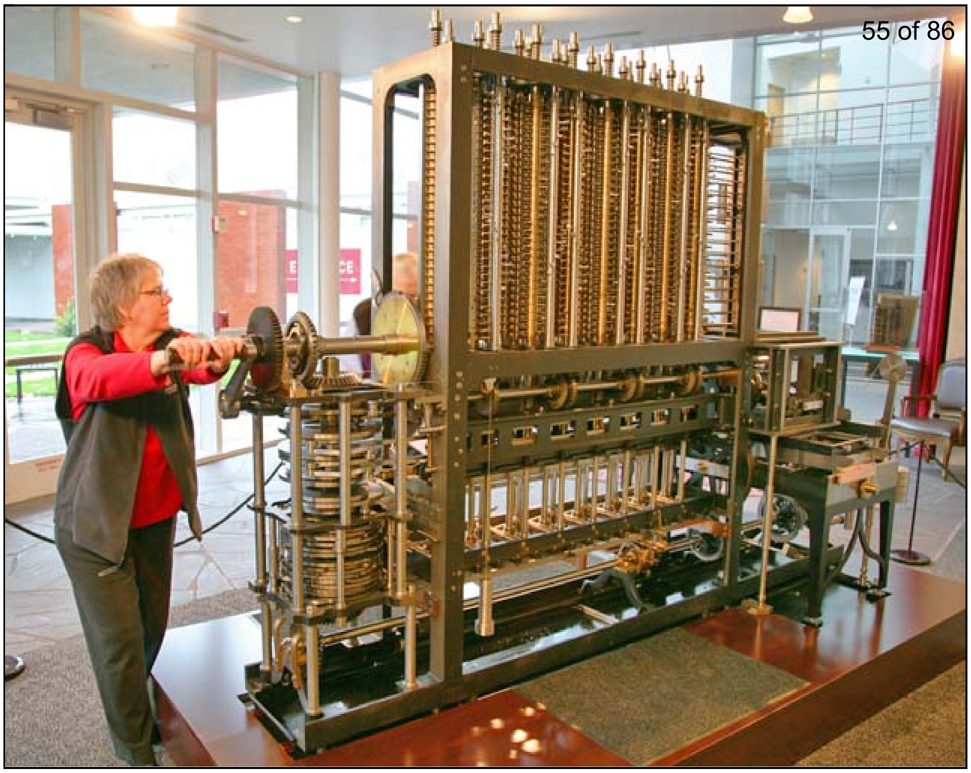
Defining Computers (cont.)

- Later, mechanical computers were designed, such as the difference engine, pioneered by Charles Babbage (1791–1871).
- This had a rather limited set of instructions it could carry out.

Defining Computers (cont.)

- Later, mechanical computers were designed, such as the difference engine, pioneered by Charles Babbage (1791–1871).
- This had a rather limited set of instructions it could carry out.

- There are seven columns of gears that can each record a decimal number. Use a crank and numbers would be added from one column of gears to the next.



Defining Computers (cont.)

- Later, mechanical computers were designed, such as the difference engine, pioneered by Charles Babbage (1791–1871).
- This had a rather limited set of instructions it could carry out.
- There are seven columns of gears that can each record a decimal number. Use a crank and numbers would be added from one column of gears to the next.
    - You could use the difference engine to compute the value of a 7th degree polynomial, with 31 digits of precision.

Defining Computers (cont.)

- Later, mechanical computers were designed, such as the difference engine, pioneered by Charles Babbage (1791–1871).
- This had a rather limited set of instructions it could carry out.
- There are seven columns of gears that can each record a decimal number. Use a crank and numbers would be added from one column of gears to the next.
    - You could use the difference engine to compute the value of a 7th degree polynomial, with 31 digits of precision.

    - The idea is that it would be used to print entire tables of logarithms for use in navigation and other fields.

Modern Computers

Modern Computers

Today, computers are more flexible and computing devices come in an ever greater variety.

Virtual Computers

Virtual Computers

- More and more computation is carried out "in the cloud."

Virtual Computers

- More and more computation is carried out "in the cloud."
    - This means that it's moved to giant warehouses that hold racks of computer servers.

Virtual Computers

- More and more computation is carried out "in the cloud."
    - This means that it's moved to giant warehouses that hold racks of computer servers.
- An example is Amazon's AWS service.

Virtual Computers

- More and more computation is carried out "in the cloud."
    - This means that it's moved to giant warehouses that hold racks of computer servers.
- An example is Amazon's AWS service.
    - You can request a type of computer, but you don't get to see it since it's in some warehouse far away.

Virtual Computers

- More and more computation is carried out "in the cloud."
    - This means that it's moved to giant warehouses that hold racks of computer servers.
- An example is Amazon's AWS service.
    - You can request a type of computer, but you don't get to see it since it's in some warehouse far away.
- This can be very efficient when you have a lot of calculations to perform, e.g., to process large data sets.

The Python Interpreter

The Python Interpreter

- Through all this complexity, one thing remains fairly constant:

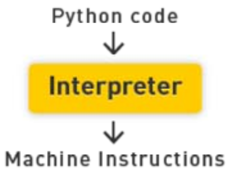
The Python Interpreter

- Through all this complexity, one thing remains fairly constant:
    - A computer is something that executes statements in a programming language.

The Python Interpreter

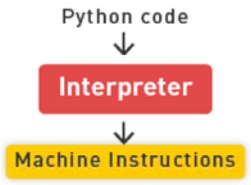
- Through all this complexity, one thing remains fairly constant:
    - A computer is something that executes statements in a programming language.
- No matter what the underlying hardware, you can expect a computer to execute Python predictably.

The Python Interpreter (cont.)



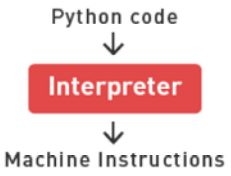
- The computer will have a program called the Python interpreter.

The Python Interpreter (cont.)



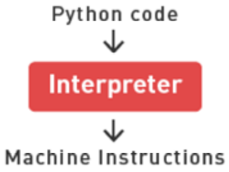
- The computer will have a program called the Python interpreter.
- This program takes Python statements and translates them into lower level instructions, that may be specific to the hardware it's running on.

The Python Interpreter (cont.)



- The computer will have a program called the Python interpreter.
- This program takes Python statements and translates them into lower level instructions, that may be specific to the hardware it's running on.
    - The interpreter ensures that whatever hardware we're running on looks the same to us as programmers.

The Python Interpreter (cont.)



Python code

Interpreter

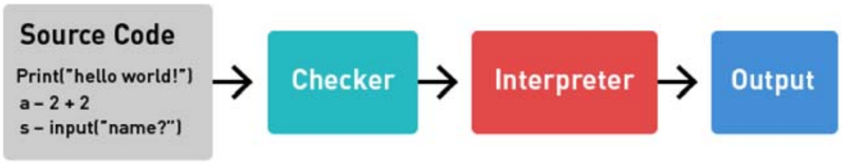
Machine Instructions

- The computer will have a program called the Python interpreter.
- This program takes Python statements and translates them into lower level instructions, that may be specific to the hardware it's running on.
    - The interpreter ensures that whatever hardware we're running on looks the same to us as programmers.
    - The interpreter is the face of the computer that we see.

Interpreted vs. Compiled

Interpreted vs. Compiled

- Interpreted (Python)



Source Code

```
Print("hello world!")
a - 2 + 2
s - input("name?")
```

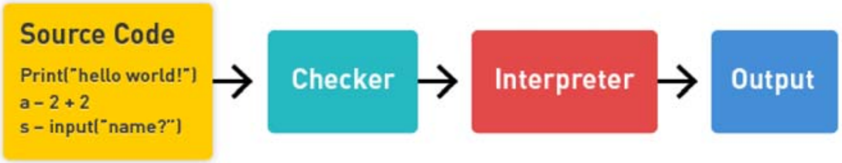
Checker

Interpreter

Output

Interpreted vs. Compiled

- Interpreted (Python)



Source Code

Print("hello world!")
a – 2 + 2
s – input("name?")

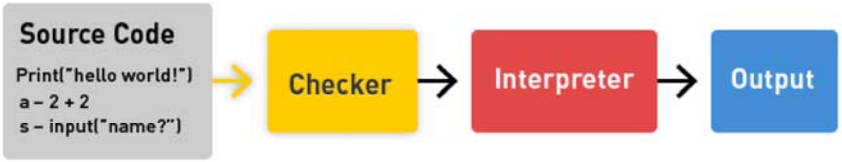
Checker

Interpreter

Output

Interpreted vs. Compiled

- Interpreted (Python)



Source Code

Print("hello world!")
a – 2 + 2
s – input("name?")

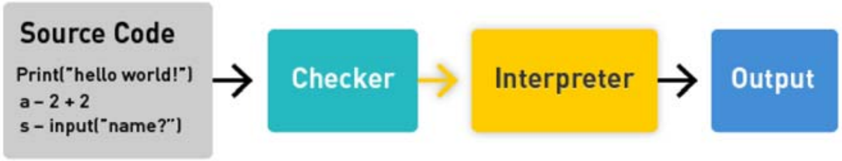
Checker

Interpreter

Output

Interpreted vs. Compiled

- Interpreted (Python)



Source Code

Print("hello world!")
a – 2 + 2
s – input("name?")

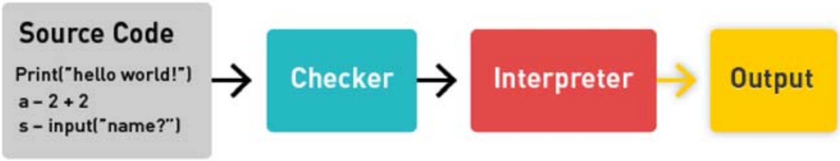
Checker

Interpreter

Output

Interpreted vs. Compiled

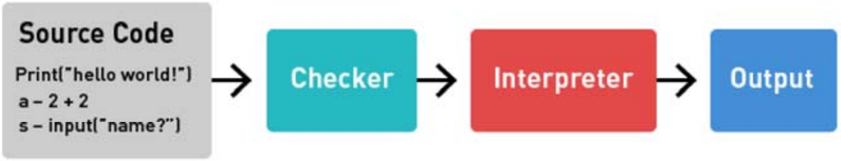
- Interpreted (Python)



```
Print("hello world!")
a – 2 + 2
s – input("name?")
```

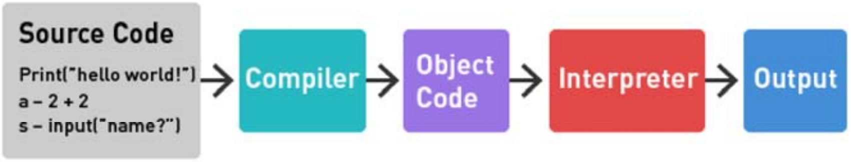
Interpreted vs. Compiled

- Interpreted (Python)



```
Print("hello world!")
a – 2 + 2
s – input("name?")
```

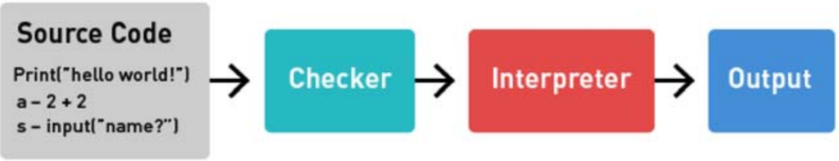
- Compiled (Java, C, etcâ�¦)



```
Print("hello world!")
a – 2 + 2
s – input("name?")
```

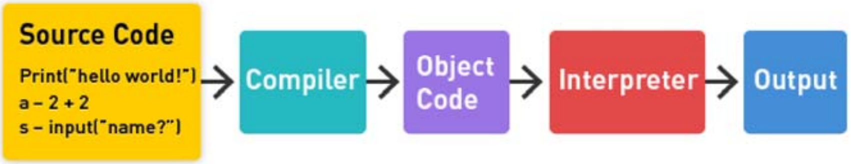
Interpreted vs. Compiled

- Interpreted (Python)



```
Print("hello world!")
a – 2 + 2
s – input("name?")
```

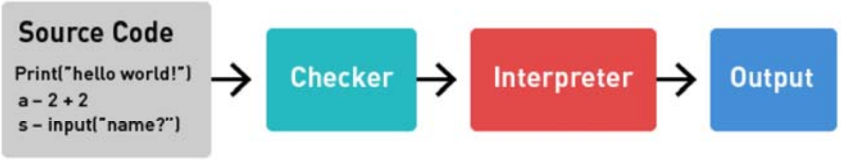
- Compiled (Java, C, etcâ�¦)



```
Print("hello world!")
a – 2 + 2
s – input("name?")
```

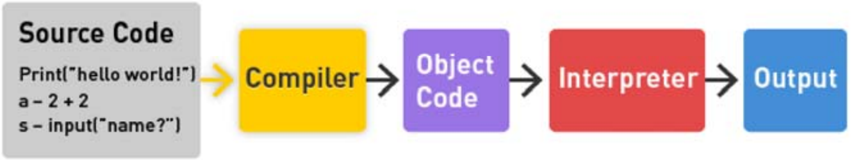
Interpreted vs. Compiled

- Interpreted (Python)



```
Print("hello world!")
a – 2 + 2
s – input("name?")
```

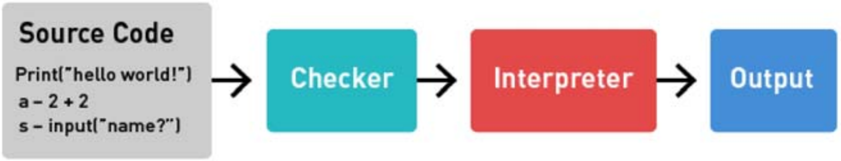
- Compiled (Java, C, etcâ�¦)



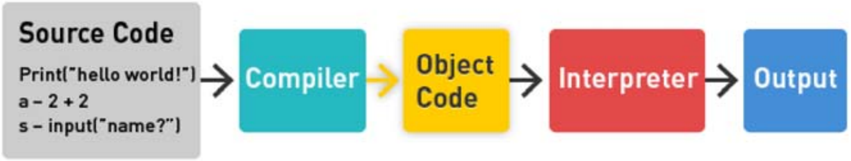
```
Print("hello world!")
a – 2 + 2
s – input("name?")
```

Interpreted vs. Compiled

- Interpreted (Python)

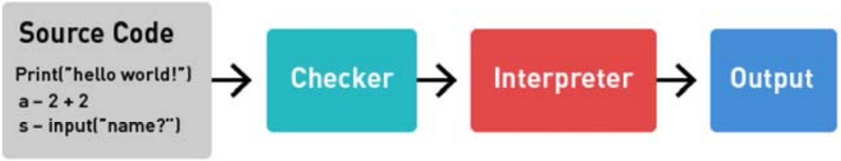


- Compiled (Java, C, etcâ�¦)

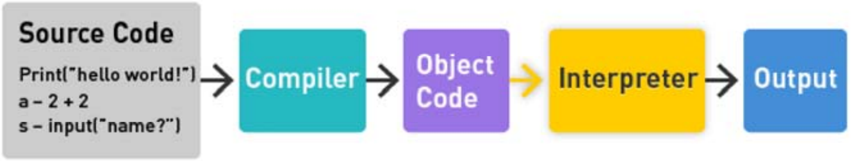


Interpreted vs. Compiled

- Interpreted (Python)

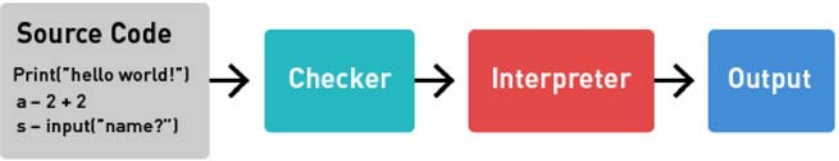


- Compiled (Java, C, etcâ�¦)



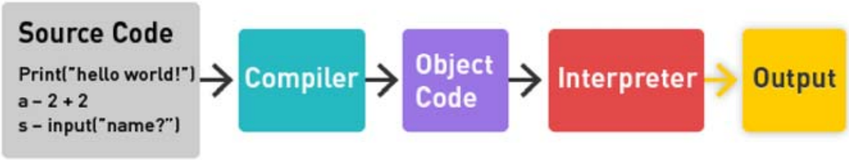
Interpreted vs. Compiled

- Interpreted (Python)



```
Print("hello world!")
a - 2 + 2
s - input("name?")
```

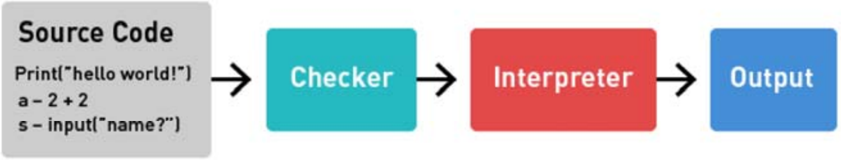
- Compiled (Java, C, etcâ�¦)



```
Print("hello world!")
a - 2 + 2
s - input("name?")
```

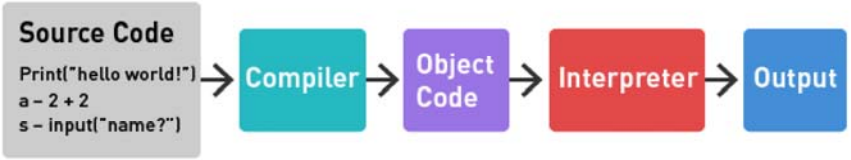
Interpreted vs. Compiled

- Interpreted (Python)



```
Print("hello world!")
a - 2 + 2
s - input("name?")
```

- Compiled (Java, C, etcâ�¦)



```
Print("hello world!")
a - 2 + 2
s - input("name?")
```