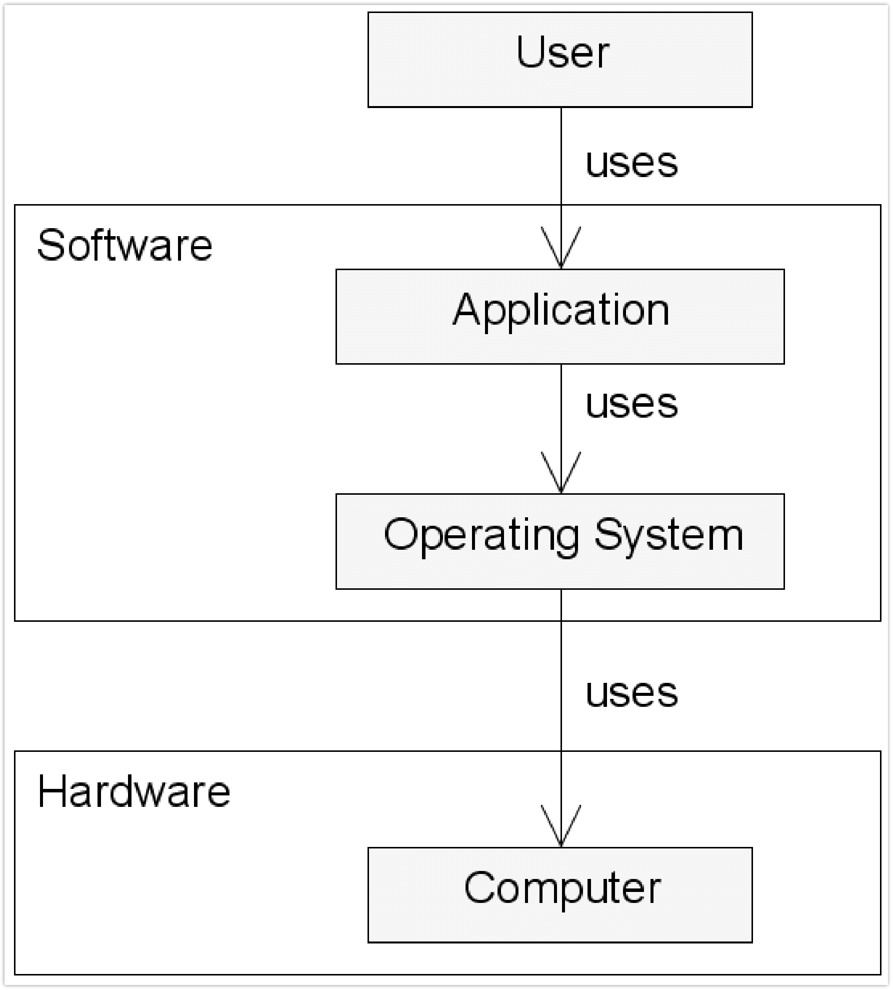
Part1 – Python Introduction

Why should you learn to write programs?

- It Guarantees You a Job
- You Get to Work From Home
- You Can Create Anything You Want
- High Income Potential
- You Understand How Software Works
- You Learn To Combine Technical Skills and Creativity
- You Develop Problem Solving Skills
- Open Your Own Business

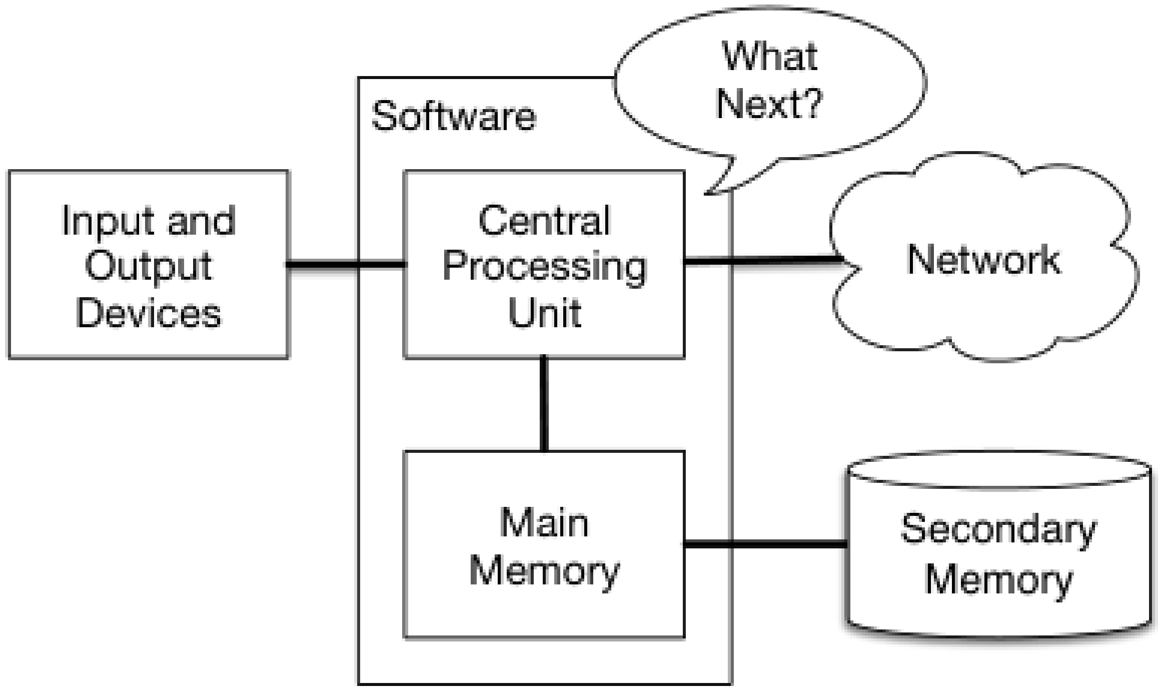


Writing programs (or programming) is a very creative and rewarding activity. You can write programs for many reasons, ranging from making your living to solving a difficult data analysis problem to having fun to helping someone else solve a problem.

Programmers add an operating system and a set of applications to the hardware and we end up with a Personal Digital Assistant that is quite helpful and capable of helping us do many different things.

Computer hardware architecture

- The Central Processing Unit (or CPU)
- The Main Memory is used to store information that the CPU needs in a hurry. The main memory is nearly as fast as the CPU. But the information stored in the main memory vanishes when the computer is turned off.
- The Secondary Memory is also used to store information, but it is much slower than the main memory. The advantage of the secondary memory is that it can store information even when there is no power to the computer. Examples of secondary memory are disk drives or flash memory
- The Input and Output Devices are simply our screen, keyboard, mouse, microphone, speaker, touchpad, etc. They are all of the ways we interact with the computer.
- Network Connection to retrieve information over a network. We can think of the network as a very slow place to store and retrieve data that might not always be “up”. So in a sense, the network is a slower and at times unreliable form of Secondary Memory.



As a programmer, your job is to use and orchestrate each of these resources to solve the problem that you need to solve and analyze the data you get from the solution. As a programmer you will mostly be "talking" to the CPU and telling it what to do next. Sometimes you will tell the CPU to use the main memory, secondary memory, network, or the input/output devices.

Benefits of Python

- **Easy To Learn:** Being an open-source platform, Python has a simple and intuitive syntax that is easy to learn and read.
- **Cross-Platform:** Python allows developers to run the code on Windows, Mac OS X, UNIX, and Linux.
- **Portable:** developer can run their code on different machines without making any further changes.
- **Extensive Library** : Python has several powerful libraries that make data analysis and visualization easy.
- **Community Support:** Python has a large and active community that supports and contributes to the development of various libraries and tools for data science.
- Python is a popular language for data science because it is easy to learn, has a large and active community, offers powerful libraries for data analysis and visualization, and has excellent machine-learning libraries.

Once you learn one programming language such as Python, you will find it much easier to learn a second programming language such as JavaScript or C++. The new programming language has very different vocabulary and grammar but the problem-solving skills will be the same across all programming languages.

Understanding programming

- you will have the skills to look at a data/information analysis problem and develop a program to solve the problem.
- you need two skills to be a programmer:
  - First, you need to know the programming language (Python) - you need to know the vocabulary and the grammar. You need to be able to spell the words in this new language properly and know how to construct well-formed "sentences" in this new language.
  - • Second, you need to "tell a story". In writing a story, you combine words and sentences to convey an idea to the reader. There is a skill and art in constructing the story, and skill in story writing is improved by doing some writing and getting some feedback. In programming, our program is the "story" and the problem you are trying to solve is the "idea".

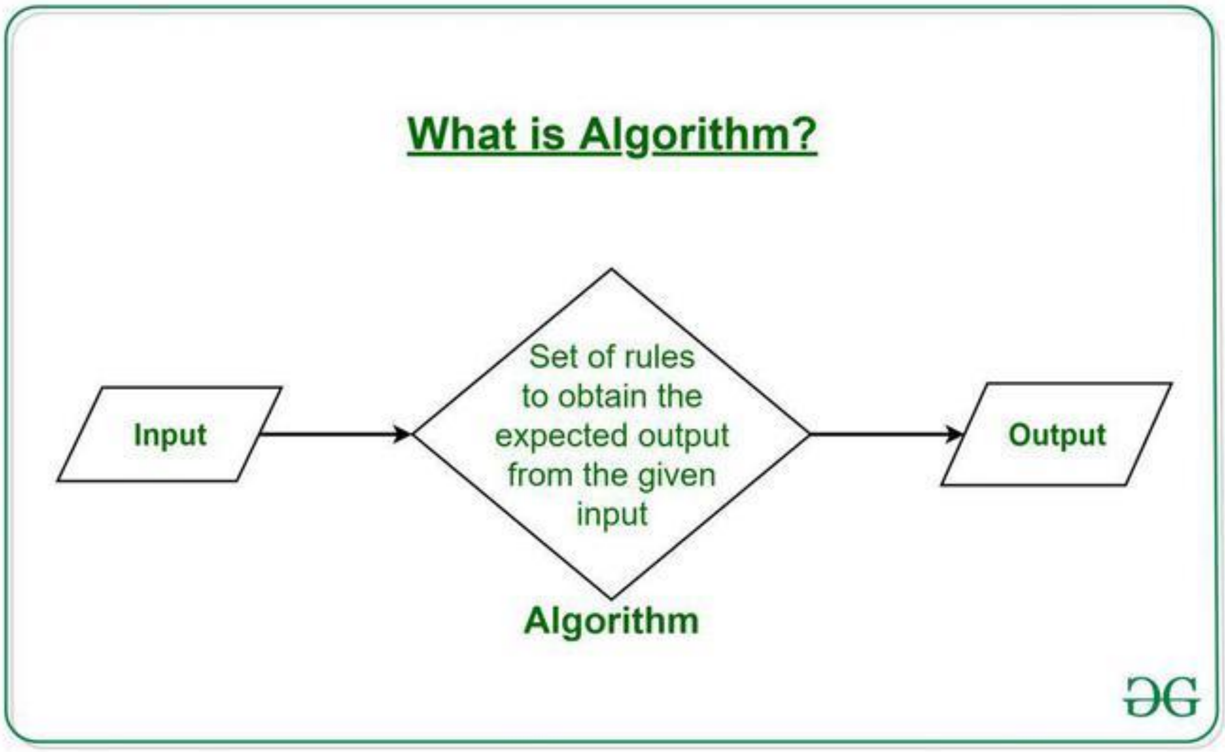
Words and sentences

1. Python vocabulary is actually pretty small. We call this “vocabulary” the “reserved words”
2. These are words that have very special meaning to Python.
3. When Python sees these words in a Python program, they have one and only one meaning to Python.
4. as you write programs you will make up your own words that have meaning to you called variables.
5. you cannot use any of Python’s reserved words as a name for a variable.

and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	
class	finally	is	return	
continue	for	lambda	try	
def	from	nonlocal	while	

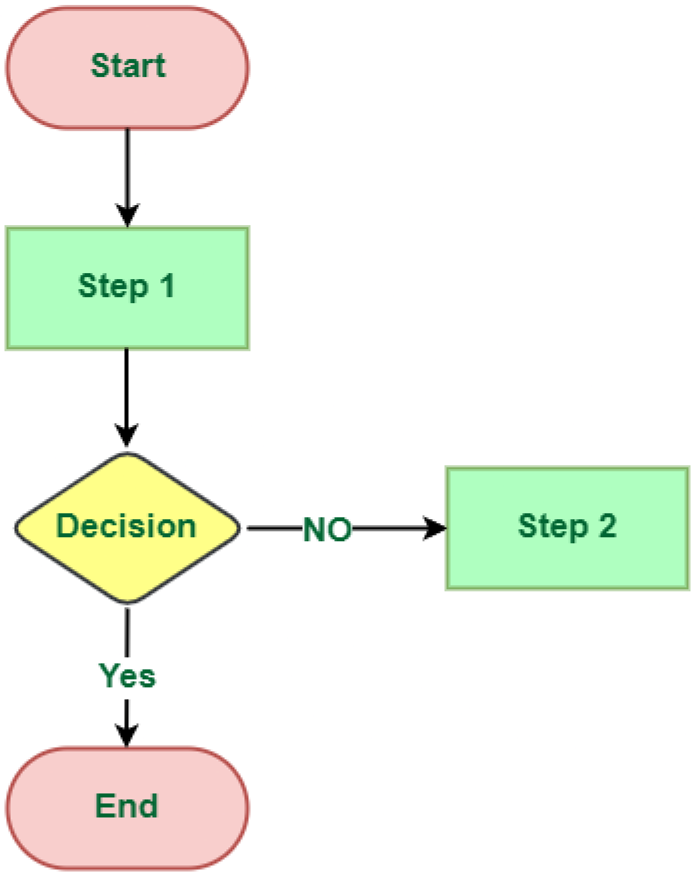
Algorithm

- A process or set of rules to be followed in calculations or other problem-solving operations
- Algorithm refers to a set of rules/instructions that step-by-step define how a work is to be executed in order to get the expected results


What is Algorithm?
Input
Set of rules to obtain the expected output from the given input
Output
Algorithm

Flowchart

- A flowchart is a graphical representation of an algorithm.
- Programmers often use it as a program-planning tool to solve a problem.
- It makes use of symbols that are connected among them to indicate the flow of information and processing.
- The process of drawing a flowchart for an algorithm is known as “flowcharting”



Conversing with Python

- ***Install Python to start a conversation with Python to test our new language skill***
- ***https://www.python.org/downloads/***
- ***https://stackoverflow.com/questions/58754860***
- ***At some point, you will be in a terminal or command window and you will type python and the Python interpreter will start executing in interactive***
- ***The >>> prompt is the Python interpreter's way of asking you, "What do you want me to do next?" Python is ready to have a conversation with you. All you have to know is how to speak the Python language.***
- ***>>> hello my name is bahman   --- syntax error***
- ***>>> print('hello my name is bahman') --- ok***
- ***The proper way to say "good-bye" to Python is to enter quit() at the interactive chevron >>> prompt.***

interpreter and compiler

- Python is a high-level language intended to be relatively straightforward for humans to read and write and for computers to read and process. Other high-level languages include Java, C++, PHP, Ruby, Basic, JS, …
- The CPU understands a language we call machine language. Machine language is very simple and frankly very tiresome to write because it is represented all in zeros and ones
- very few programmers ever write machine language.
- Instead we build various translators to allow programmers to write in high-level languages like Python or JavaScript and these translators convert the programs to machine language for actual execution by the CPU.
- Since machine language is tied to the computer hardware, machine language is not portable across different types of hardware. Programs written in high-level languages can be moved between different computers by using a different interpreter on the new machine or recompiling the code to create a machine language version of the program for the new machine.
- These programming language translators fall into two general categories: (1) interpreters and (2) compilers.
- Compiler: To translate a program written in a high-level language into a low-level language all at once, in preparation for later execution.
- Interpreter: To execute a program in a high-level language by translating it one line at a time.
- An interpreter reads the source code of the program as written by the programmer, parses the source code, and interprets the instructions on the fly. Python is an interpreter and when we are running Python interactively, we can type a line of Python (a sentence) and Python processes it immediately and is ready for us to type another line of Python.
- Even though we are typing these commands into Python one line at a time, Python is treating them as an ordered sequence of statements with later statements able to retrieve data created in earlier statements. We are writing our first simple paragraph with four sentences in a logical and meaningful order.
- A compiler needs to be handed the entire program in a file, and then it runs a process to translate the high-level source code into machine language and then the compiler puts the resulting machine language into a file for later execution.
- If you have a Windows system, often these executable machine language programs have a suffix of ".exe" or ".dll" which stand for "executable" and "dynamic link library" respectively. In Linux and Macintosh, there is no suffix that uniquely marks a file as executable.

- The interpreter reads the source code one line at a time and executes it

Python Interpreter

- The Python interpreter is written in a high-level language called “C”. You can look at the actual source code for the Python interpreter by going to www.python.org and working your way to their source code.
- Python is a program itself and it is compiled into machine code. When you installed Python on your computer (or the vendor installed it), you copied a machine-code copy of the translated Python program onto your system. In Windows, the executable machine code for Python itself is likely in a file with a name like: C:\Python35\python.exe

Writing a program

- Open a note pad
- Write code in
    - print('Hello world!')
- Save file as hello.py
- Open cmd in folder: python hello.py
- Next: write code in VSCode

What is a program

- sequence of Python statements that have been written to do something.
- When we want to write a program, we use a text editor to write the Python instructions into a file, which is called a script. By convention, Python scripts have names that end with .py.
- our simple hello.py script is a program. It is not particularly useful, but in the strictest definition, it is a Python program.
- It might be easiest to understand what a program is by thinking about a problem that a program might be built to solve, and then looking at a program that would solve that problem.

The building blocks of programs

- **Input:** Get data from the “outside world”. reading data from a file, or even some kind of sensor like a microphone or GPS. or from the user typing data on the keyboard.
- **Output:** Display the results of the program on a screen or store them in a file or perhaps write them to a device like a speaker to play music or speak text.

- **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.

Program Errors

- **Syntax errors** These are the first errors you will make and the easiest to fix. A syntax error means that you have violated the "grammar" rules of Python.
- **Logic errors** When your program has good syntax but there is a mistake in the order of the statements or perhaps a mistake in how the statements relate to one another.
- **Semantic errors** The program is perfectly correct but it does not do what you intended for it to do.

Chapter 2 - Variables, expressions, and statements

Values and types

- A value is one of the basic things a program works with, like a letter or a number. The values we have seen so far are 1, 2, and "Hello, World!"
- These values belong to different types: 2 is an integer, and "Hello, World!" is a string, so called because it contains a "string" of letters.
- You (and the interpreter) can identify strings because they are enclosed in quotation marks.
- If you are not sure what type a value has, the interpreter can tell you.

```
>>> type('Hello, World!')
<class 'str'>
>>> type(17)
<class 'int'>
```

Values and types

- strings belong to the type str and integers belong to the type int. Less obviously, numbers with a decimal point belong to a type called float, because these numbers are represented in a format called floating point.

```
>>> type(3.2)
<class 'float'>
```

```
>>> type('17')
<class 'str'>
>>> type('3.2')
<class 'str'>
```

Variables

- A variable is a name that refers to a value.
- An assignment statement creates new variables and gives them values

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.1415926535897931
```

- Variable names should be meaningful and document what the variable is used for.
- They can contain both letters and numbers, but they cannot start with a number
- it is a good idea to begin variable names with a lowercase letter
- The underscore character (_) can appear in a name. It is often used in names with multiple words, such as my_name or airspeed_of_unladen_swallow.
- It turns out that class is one of Python's keywords. The interpreter uses keywords to recognize the structure of the program, and they cannot be used as variable names.

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

Statements

- A statement is a unit of code that the Python interpreter can execute. For example:
  - print being an expression statement and assignment.
- A script usually contains a sequence of statements.
- The assignment statement produces no output.

Operators and operands

- Operators are special symbols that represent computations like addition and multiplication.Ex: +, -, *, /,**
- The values the operator is applied to are called operands.

```
20+32    hour-1    hour*60+minute    minute/60    5**2    (5+9)*(15-7)
```

Expressions

- An expression is a combination of values, variables, and operators.
- A value all by itself is considered an expression, and so is a variable, so the following are all legal expressions (assuming that the variable x has been assigned a value)

```
17
x
x + 17
```

- If you type an expression in interactive mode, the interpreter evaluates it and displays the result.

```
>>> 1 + 1
2
```

- But in a script, an expression all by itself doesn't do anything!

Order of operations

- 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:
  - Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want.

```
(1+1)**(5-2)
```

  - Exponentiation has the next highest precedence

```
2**1+1
```

  - Multiplication and Division have the same precedence, which is higher than Addition and Subtraction, which also have the same precedence.

```
2*3-1
```

  - Operators with the same precedence are evaluated from left to right.

```
5-3-1
```

Modulus operator

- The modulus operator works on integers and yields the remainder when the firstoperand is divided by the second. In Python, the modulus operator is a percent sign (%). The syntax is the same as for other operators:

```
>>> quotient = 7 // 3
>>> print(quotient)
2
>>> remainder = 7 % 3
>>> print(remainder)
1
```

String operations

- The + operator works with strings, but it is not addition in the mathematical sense.
- Instead it performs concatenation, which means joining the strings by linking them end to end.

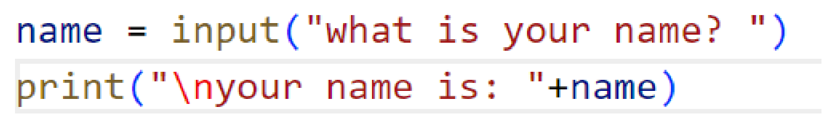
```
>>> first = 10
>>> second = 15
>>> print(first+second)
25
>>> first = '100'
>>> second = '150'
>>> print(first + second)
100150
```

Asking the user for input

- Take the value for a variable from the user via their keyboard.
- Python provides a built-in function called input that gets input from the keyboard.
- When this function is called, the program stops and waits for the user to type something.

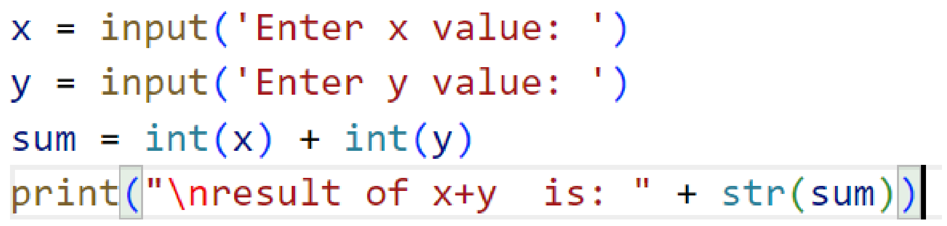
```
name = input()
print("your name is: "+name)
```

- print a prompt telling the user what to input.



```python
name = input("what is your name? ")
print("\nyour name is: "+name)
```

- The sequence \n at the end of the prompt represents a newline, which is a special character that causes a line break.
- you can try to convert the return value to int using the int() function



```python
x = input('Enter x value: ')
y = input('Enter y value: ')
sum = int(x) + int(y)
print("\nresult of x+y  is: " + str(sum))
```

The strings in the print statements are enclosed in quotes. Single quotes and double quotes do the same thing; most people use single quotes except in cases like this where a single quote (which is also an apostrophe) appears in the string.

Comments

- add notes to your programs to explain in natural language what the program is doing.
- Comments in Python they start with the # symbol
- Everything from the \# to the end of the line is ignored; it has no effect on the program.

```python
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60

percentage = (minute * 100) / 60    # percentage of an hour

v = 5   # assign 5 to v

v = 5   # velocity in meters/second.
```

Chapter 3- Conditional execution

Boolean expressions

- A boolean expression is an expression that is either true or false.

```
>>> 5 == 5
True
>>> 5 == 6
False
{}
```

- True and False are special values that belong to the class bool; they are not strings.

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

- comparison operators;

```
x != y               # x is not equal to y
x > y                # x is greater than y
x < y                # x is less than y
x >= y               # x is greater than or equal to y
x <= y               # x is less than or equal to y
x is y               # x is the same as y
x is not y           # x is not the same as y
```

- A common error is to use a single equal sign (=) instead of a double equal sign (==). Remember that = is an assignment operator and == is a comparison operator.

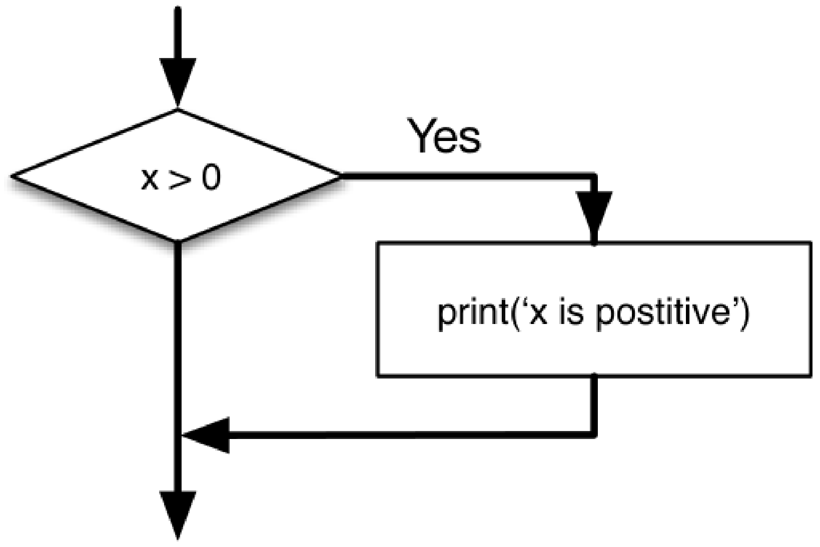
Logical operators

- There are three logical operators: and, or, and not.
- x > 0 and x < 10: is true only if x is greater than 0 and less than 10.
- Any nonzero number is interpreted as "true."

```
>>> 17 and True
True
```

Conditional execution

- Conditional statements give us the ability to check conditions and change the behavior of the program. The simplest form is the if statement:



```
if x > 0 :
    print('x is positive')
```

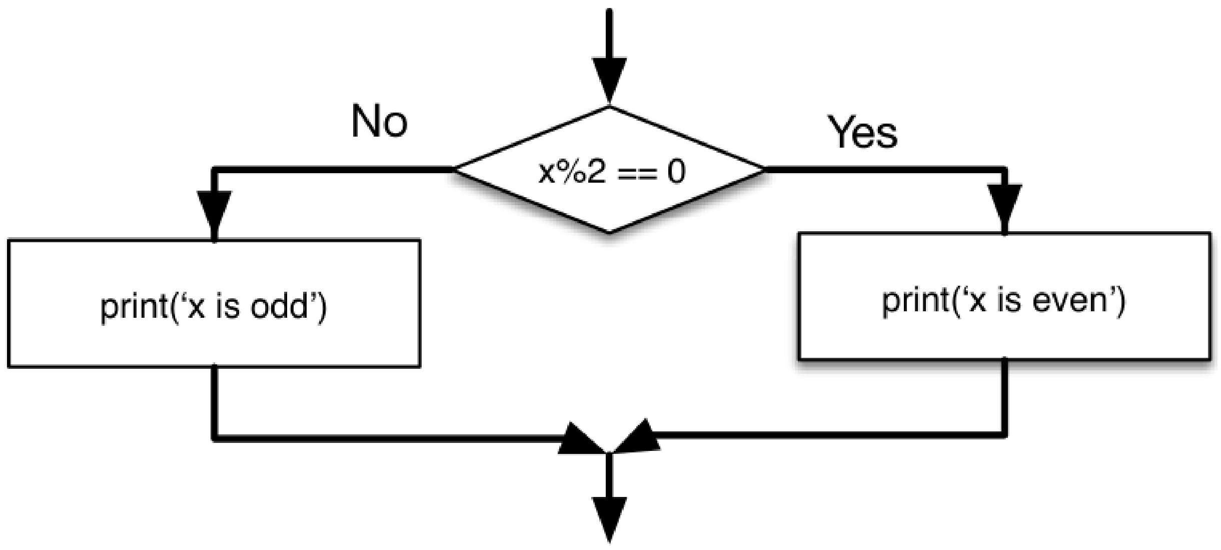
- The boolean expression after the if statement is called the condition.
- We end the if statement with a colon character (:)
- the line(s) after the if statement are indented.

- There is no limit on the number of statements that can appear in the body, but there must be at least one.
- it is useful to have a body with no statements (usually as a place holder for code you haven't written yet). In that case, you can use the pass statement, which does nothing

```
if x < 0 :
    pass          # need to handle negative values!
```

ALTERNATIVE EXECUTION

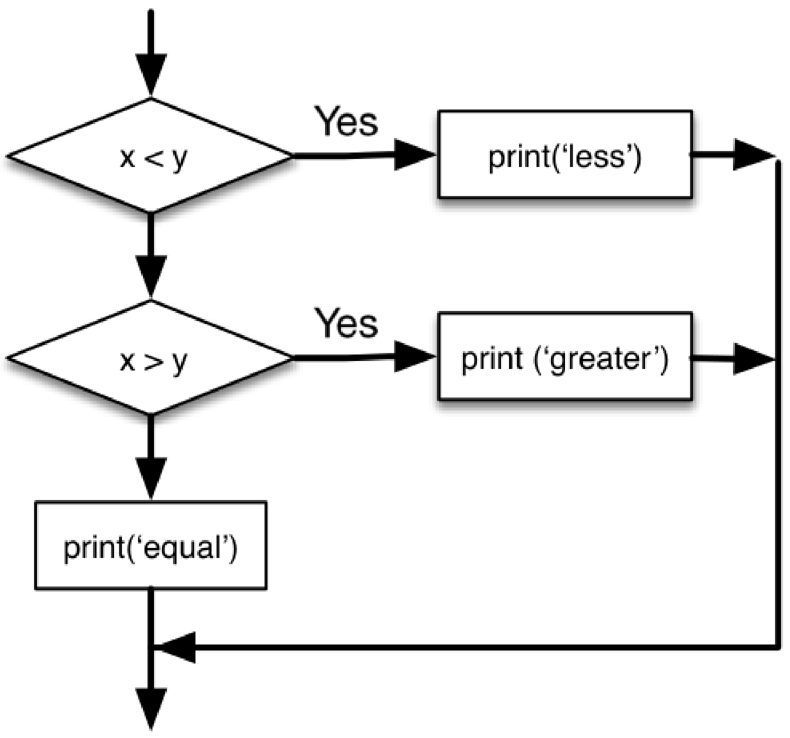
- there are two possibilities and the condition determines which one gets executed.



```
if x%2 == 0 :
    print('x is even')
else :
    print('x is odd')
```

Chained conditionals

- there are more than two possibilities and we need more than two branches.

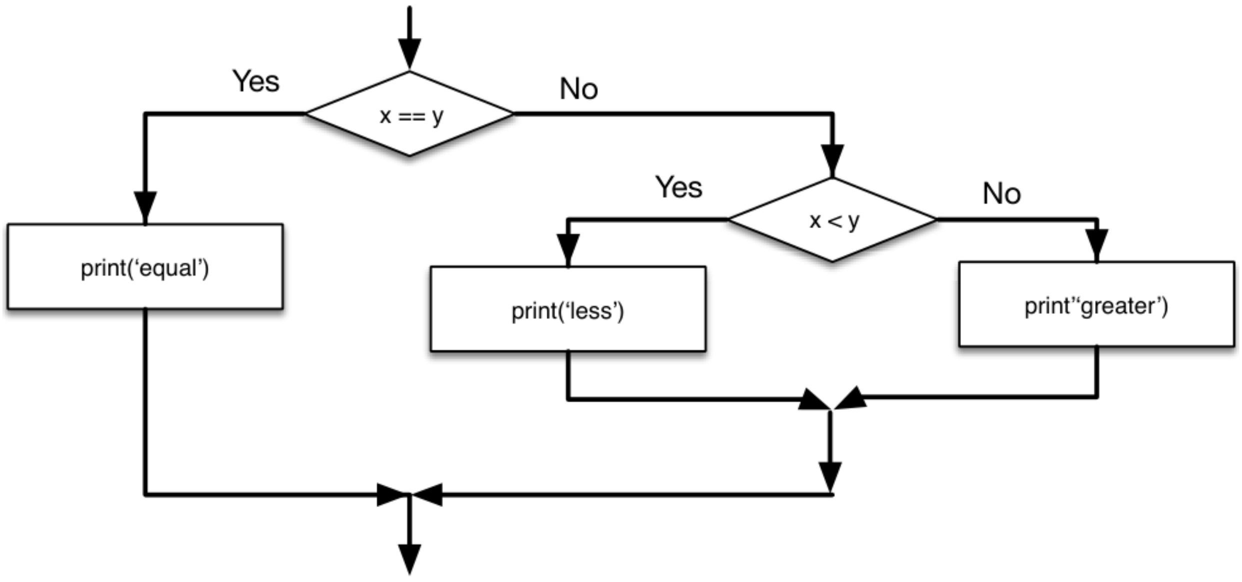


```
if x < y:
    print('x is less than y')
elif x > y:
    print('x is greater than y')
else:
    print('x and y are equal')
```

- elif is an abbreviation of "else if."
- Each condition is checked in order. If the first is false, the next is checked, and so on.
- If one of them is true, the corresponding branch executes, and the statement ends. Even if more than one condition is true, only the first true branch executes.

Nested conditionals

- One conditional can also be nested within another.



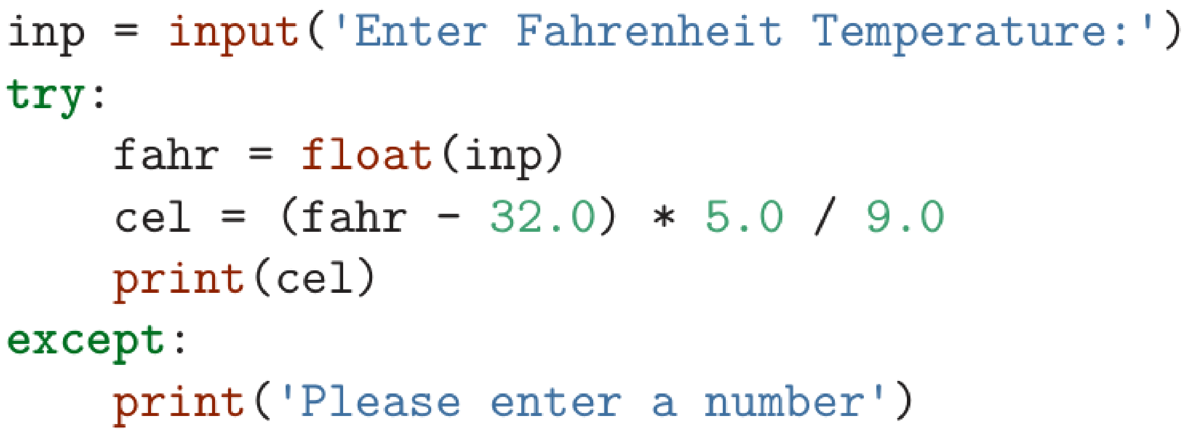
```
if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y')
    else:
        print('x is greater than y')
```

```
if 0 < x:
    if x < 10:
        print('x is a positive single-digit number.')
```

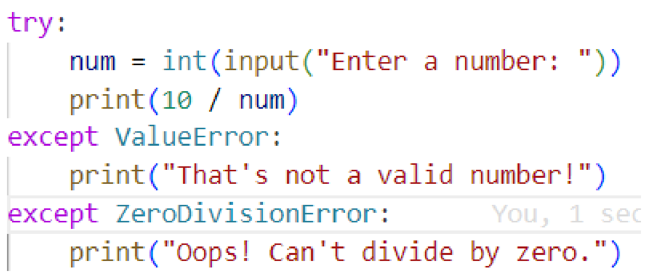
Catching exceptions using try and except

- we use **try** and **except** blocks to handle errors.
- This allows our program to continue running even when it encounters an error.
- **Multiple Exceptions:** You can handle different types of errors separately.

```
try:
    # Code that might cause an error
except ErrorType:
    # Code to run if there is an error
```



```python
inp = input('Enter Fahrenheit Temperature:')
try:
    fahr = float(inp)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print(cel)
except:
    print('Please enter a number')
```



```python
try:
    num = int(input("Enter a number: "))
    print(10 / num)
except ValueError:
    print("That's not a valid number!")
except ZeroDivisionError:        You, 1 sec
    print("Oops! Can't divide by zero.")
```

Short-circuit evaluation of logical expressions

- Python evaluates the expression from left to right
- 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.
- While this may seem like a fine point, the short-circuit behavior leads to a clever technique called the guardian pattern.
- we say that y != 0 acts as a guard to insure that we only execute (x/y) if y is non-zero.

```
>>> x = 1
>>> y = 0
>>> x >= 2 and y != 0 and (x/y) > 2
False
>>> x = 6
>>> y = 0
>>> x >= 2 and y != 0 and (x/y) > 2
False
>>> x >= 2 and (x/y) > 2 and y != 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

References

- C. R. Severance. Python for Everybody: Exploring Data in Python 3. CreateSpace Independent Publishing, 2016.
- 10 Reasons Why You Should Learn Programming, potomac.edu
- Difference Between Algorithm and Flowchart, geeksforgeeks.org
- Python for Data Science, geeksforgeeks.org