**Welcome to the course!**

This course is designed to teach you the foundations of programming in Python. We’re excited to join you on this journey as you learn one of the most-in-demand job skills in IT today. In the U.S. alone, according to Burning Glass data from May 2019, there were ~530K job openings in 2018 asking for Python skills.

**Course prerequisites**

This course requires no previous knowledge of programming. Familiarity with basic IT concepts, like operating systems, files and processes, networking and data management will be required in further courses. For learners with no IT background at all, we recommend taking the [IT Support Professional Certificate 101](https://www.coursera.org/specializations/google-it-support).

In Python, text in between quotes -- either single or double quotes -- is a string data type. An integer is a whole number, without a fraction, while a float is a real number that can contain a fractional part. For example, 1, 7, 342 are all integers, while 5.3, 3.14159 and 6.0 are all floats. When attempting to mix incompatible data types, you may encounter a **TypeError**. You can always check the data type of something using the type() function.

As we saw earlier in the video, some data types can be mixed and matched due to implicit conversion. Implicit conversion is where the interpreter helps us out and automatically converts one data type into another, without having to explicitly tell it to do so.

By contrast, explicit conversion is where we manually convert from one data type to another by calling the relevant function for the data type we want to convert to. We used this in our video example when we wanted to print a number alongside some text. Before we could do that, we needed to call the str() function to convert the number into a string. Once the number was explicitly converted to a string, we could join it with the rest of our textual string and print the result.

# Study Guide: Expressions and Variables

This study guide provides a quick-reference summary of what you learned in this lesson and serves as a guide for the upcoming practice quiz.

In the Expressions and Variables segment, you learned about expressions, variables, and the data types: string, integer, and float. You learned how to convert a value from one data type to another and you learned how to resolve a few common errors in Python.

# Terms

* **expression** - a combination of numbers, symbols, or other values that produce a result when evaluated
* **data types** - classes of data (e.g., string, int, float, Boolean, etc.), which include the properties and behaviors of instances of the data type (variables)
* **variable** - an instance of a data type class, represented by a unique name within the code, that stores changeable values of the specific data type
* **implicit conversion** - when the Python interpreter automatically converts one data type to another
* **explicit conversion** - when code is written to manually convert one data type to another using a data type conversion function:
  + **str()** - converts a value (often numeric) to a **string** data type
  + **int()** - converts a value (usually a float) to an **integer** data type
  + **float()** - converts a value (usually an integer) to a **float** data type

# Coding skills

**Skill Group 1**

* Use the assignment operator **=** to assign values to variables
* Use basic arithmetic operators with variables to create expressions
* Use explicit conversion to change a data type from float to string

1

2

3

4

5

6

7

8

9

10

11

12

# The following lines assign the variable to the left of the =

# assignment operator with the values and arithmetic expressions

# on the right side of the = assignment operator.

hotel\_room = 100

tax = hotel\_room \* 0.08

total = hotel\_room + tax

room\_guests = 4

share\_per\_person = total/room\_guests

# This line outputs the result of the final calculation stored

# in the variable "share\_per\_person"

print("Each person needs to pay: " + str(share\_per\_person)) # change a data type





RunReset

**Skill Group 2**

* Output multiple string variables on a single line to form a sentence
* Use the plus (+) connector or a comma to connect strings in a print() function
* Create spaces between variables in  a print() function

1

2

3

4

5

6

7

8

9

10

11

12

13

14

# The following 5 lines assign strings to a list of variables.

salutation = "Dr."

first\_name = "Prisha"

middle\_name = "Jai"

last\_name = "Agarwal"

suffix = "Ph.D."

print(salutation + " " + first\_name + " " + middle\_name + " " + last\_name + ", " + suffix)

# The comma as a string ", " adds the conventional use of a comma plus a

# space to separate the last name from the suffix.

# Alternatively, you could use commas in place of the + connector:

print(salutation, first\_name, middle\_name, last\_name,",", suffix)

# However, you will find that this produces a space before a comma within a string.





RunReset

**Skill Group 3**

* Resolve TypeError caused by a data type mismatch issue
* Use an explicit conversion function

1

2

3

4

5

6

7

8

print("5 \* 3 = " + str(5\*3))

# Resolution:

# print("5 \* 3 = " + str(5\*3))

#

# To avoid a type error between the string and the integer within the

# print() function, you can make an explicit data type conversion by

# using the str() function to convert the integer to a string.





RunReset

**Skill Group 4**

* Resolve a ZeroDivisionError caused by an attempt to divide by 0

1

2

3

4

5

6

7

8

9

10

11

numerator = 7

denominator = 1  # Possible resolution: Change the denominator value

result = numerator / denominator

print(result)

# One possible assumption for a number divided by zero error might

# include the issue of a null value as a denominator (could happen when

# using a loop to iterate over values in a database). In such cases, the

# desired outcome may be to leave the numerator value intact. The

# numerator value can be preserved by reassigning the denominator with

# the integer value of 1. The result would then equal the numerator.





RunReset

# Python practice information

For additional Python practice, the following links will take you to several popular online interpreters and codepads:

* [Welcome to Python](https://www.python.org/shell/)
* [Online Python Interpreter](https://www.onlinegdb.com/online_python_interpreter)
* [Create a new Repl](https://repl.it/languages/python3)
* [Online Python-3 Compiler (Interpreter)](https://www.tutorialspoint.com/execute_python3_online.php)
* [Compile Python 3 Online](https://rextester.com/l/python3_online_compiler)
* [Your Python Trinket](https://trinket.io/python3)

# Uses for Automation

Scripts can be used for automating specific tasks. Automation is used to replace a repetitive manual step with one that happens automatically. Humans are fallible. They can become tired, make mistakes, fail to follow instructions, be inconsistent in their job performance, and more. In contrast, automated processes complete instructions exactly as coded, in a consistent manner. They can run 24 hours a day, everyday, without tiring. For many tasks that are appropriate for automation, it can be more cost effective to use automation than human labor.

## ****Appropriate uses for automation include:****

* The automatic timing and regulation of traffic lights
* A repetitive task that is at high risk for human error
* Sending commands to a computer
* Detecting and removing duplicates of data
* Sending automated emails that are personalized by pulling individual names from a database and plugging them into the email
* Updating a large number of file permissions
* Reporting on system data, like disk or memory usage
* Installing software
* Generating reports
* Deploying a file or a computer program to all computers on a company network
* Using a configuration management system to deploy software patches, after a human has designed the system
* Populating an e-commerce site with products
* Setting the home directory and access permissions for users

## ****Automation is not always an appropriate or complete solution****

Automation cannot perform all human work. Tasks that call for human creativity, social connection, psychology, flexibility, ingenuity, evaluation, and/or complex analytic work are not good candidates for full automation. Sometimes automation can be used to perform one or more subtasks of a larger set of tasks – but – human intervention is required to complete the tasks. The following are some examples of tasks that cannot or should not be **fully** automated:

* Items that require human evaluation and analytic skills:
  + Designing a configuration management system
  + Investigating and troubleshooting all end user problems
  + Writing a computer program
  + Building a new startup business
* Items that require human creativity and/or an eye for aesthetic qualities:
  + Designing an attractive webpage (AI can do this, but simple automation cannot)
  + Wedding photography
  + Haircuts and styling
* Items that cannot be automated due to basic physics:
  + Troubleshooting or repairing machines that cannot power on or boot up
* Items that need human interaction, psychology, and/or evaluation skills:
  + Interviewing and hiring new employees
  + Customer service (chat bots cannot address every customer service need)
* Items that should not be fully automated due to costs and safety:
  + Grocery store checkout process, including bagging groceries
  + Tasks that are less expensive to perform manually

## ****Artificial Intelligence****

It is important to understand that basic automation is not the same as artificial intelligence. Automation is used to explicitly instruct a machine on how to perform a task. Artificial intelligence (AI) involves training a computing machine to perform more complex tasks through a process called machine learning. This process prepares the AI software to perform new tasks without a human needing to program explicit instructions for each task. Although AI is often used for automating human tasks, AI automation is much more complex than basic automation.

# Study Guide: Introduction to Programming

Your first practice quiz is coming up soon. This handy study guide should help you prepare for that quiz. The practice quizzes do not count towards your grade in this course. Practice quizzes are opportunities for you to check your understanding of the materials before you take the graded assessments at the end of each module.

# Key Terms

* **Programming code** - Programming code is a set of written computer instructions, guided by rules, using a computer programming language. It might help to think of the computer instructions as a detailed, step-by-step recipe for performing tasks. The instructions tell computers and machines how to perform an action. Programming code may also be referred to as source code or scripts.
* **Programming languages** - Programming languages are similar to human spoken languages in that they both use syntax and semantics. Programming languages are used to write computer programs.  Some common programming languages include Python, Java, C, C++, C#, and R.
* **Syntax** - Syntax is a set of rules for how statements are constructed in both human and computer languages. Programming syntax includes rules for the order of elements in programming instructions, as well as the use of special characters and their placements in statements. This concept is similar to the syntax rules for grammar and punctuation in human language.
* **Semantics** - Semantics refers to the intended meaning or effect of statements, or collections of words, in both human and computer languages. Semantic errors are also referred to as logical errors.
* **Computer program** - A computer program is a step-by-step list of instructions that a computer follows to reach an intended goal. It is important to be clear and precise about the actions a computer program is supposed to perform because computers will do exactly what they are instructed to do. Computer programs can be long, complex, and accomplish a variety of tasks. They are often developed by computer programmers and software engineers, but anyone can learn to create them. Computer programs may involve a structured development cycle. They can be written in a wide variety of programming languages, such as Python, Java, C++,  R, and more. The completed format of a program is often a single executable file.
* **Script** - Scripts are usually shorter and less complex than computer programs. Scripts are often used to automate specific tasks. However, they can be used for complex tasks if needed. Scripts are often written by IT professionals, but anyone can learn to write scripts. Scripts have a shorter, less structured development cycle as compared to the development of complex computer programs and software. Scripts can be written in a variety of programming languages, like Python, Javascript, Ruby, Bash, and more. Some scripting languages are interpreted languages and are only compatible with certain platforms.
* **Automation** - Automation is used to replace a repetitive manual step with one that happens automatically.
* **Output** - Output is the end result of a task performed by a function or computer program. Output can include a single value, a report, entries into a database, and more.
* **Input** - Input is information that is provided to a program by the end user. Input can be text, voice, images, biometrics, and more.
* **Functions** - A function is a reusable block of code that performs a specific task.
* **Variables** - Variables are used to temporarily store changeable values in programming code.

# Python Resources

## More About Python

### Using Python on your own

The best way to learn any programming language is to practice it on your own as much as you can. If you have Python installed on your computer, you can execute the interpreter by running the python3 command (or just python on Windows), and you can close it by typing exit() or Ctrl-D.

If you don’t already have Python installed on your machine, that’s alright. We’ll explain how to install it in an upcoming course.

### Python practice resources

In the meantime, you can still practice by using one of the many online Python interpreters or codepads available online. There’s not much difference between an interpreter and a codepad. An interpreter is more interactive than a codepad, but they both let you execute code and see the results.

Below, you’ll find links to some of the most popular online interpreters and codepads. Give them a go to find your favorite.

* <https://www.python.org/shell/>
* <https://www.onlinegdb.com/online_python_interpreter>
* <https://repl.it/languages/python3>
* <https://www.tutorialspoint.com/execute_python3_online.php>
* <https://rextester.com/l/python3_online_compiler>
* <https://trinket.io/python3>

### Additional Python resources

While this course will give you information about how Python works and how to write scripts in Python, you’ll likely want to find out more about specific parts of the language. Here are some great ways to help you find additional info:

* Read the [official Python documentation](https://docs.python.org/3/).
* Search for answers or ask a question on [Stack Overflow](https://stackoverflow.com/).
* Subscribe to the Python [tutor](https://mail.python.org/mailman/listinfo/tutor) mailing list, where you can ask questions and collaborate with other Python learners.
* Subscribe to the [Python-announce](https://mail.python.org/mailman/listinfo/python-announce-list) mailing list to read about the latest updates in the language.

### Python history and current status

Python was released almost 30 years ago and has a rich history. You can read more about it on the [History of Python](https://en.wikipedia.org/wiki/History_of_Python) Wikipedia page or in the section on the [history of the software](https://docs.python.org/3.0/license.html) from the official Python documentation.

Python has recently been called the fastest growing programming language. If you're interested in why this is and how it’s measured, you can find out more in these articles:

* [The Incredible Growth of Python](https://stackoverflow.blog/2017/09/06/incredible-growth-python/) (Stack Overflow)
* [Why is Python Growing So Quickly - Future Trends](https://www.netguru.com/blog/why-python-is-growing-so-quickly-future-trends) (Netguru)
* [By the numbers: Python community trends in 2017/2018](https://opensource.com/article/18/5/numbers-python-community-trends) (Opensource.com)
* [Developer Survey Results 2018](https://insights.stackoverflow.com/survey/2018#technology) (Stack Overflow)

# A Note on Syntax and Code Blocks

When writing code, using correct syntax is critical. Even a small typo, like a missing parenthesis bracket or an extra comma, can cause a syntax error and the code won't execute at all. If your code results in an error or an exception, pay close attention to syntax and watch out for minor mistakes. A single wrong character could take hours to identify in long code so it is important to be mindful of syntax when writing code.

## ****Common syntax errors:****

* Misspellings
* Incorrect indentations
* Missing or incorrect key characters:
  + Bracket types - ( curved ), [ square ], { curly }
  + Quote types - "straight-double" or 'straight-single', “curly-double” or ‘curly-single’
  + Block introduction characters, like colons - :
* Data type mismatches
* Missing, incorrectly used, or misplaced Python reserved words
* Using the wrong case (uppercase/lowercase) - Python is a case-sensitive language

If your syntax is correct, but the script has unexpected behavior or output, this may be due to a semantic problem. Syntax is like the vocabulary, grammar, spelling, and punctuation of code. Semantics are the meaning and logic of coded statements. It is possible to have syntactically correct code that runs successfully, but doesn't do what we want it to do.

## ****Common semantic errors:****

* Creating functional code, but getting unintentional output
* Poor logic structures in the design of the code

When working with the code blocks in exercises for this course, be mindful of syntax and semantic (logic) errors, along with the overall result of your code. Just because you fixed an error doesn't mean that the code will have the desired effect when it runs! Once you’ve fixed an error in your code, don't forget to click Run to check your work.

# Study Guide: Introduction to Python

This study guide provides a quick-reference summary of what you learned in this lesson and serves as a guide for the upcoming practice quiz.

In this segment, you learned that Python is a general purpose programming language that is commonly used for scripting and automation, as well as to develop a wide variety of applications. Python is compatible with most operating systems, including Windows, Linux, and Mac OS, and is updated every few years. Python can also run on a variety of machines, such as servers, workstations, PCs, mobile devices, IoT, and more.

Python is widely used in the IT field, including IT support, system administration, web development, machine learning, data analytics, and more. Python can be used to calculate statistics, run your e-commerce site, process images, interact with web services, and do a whole host of other tasks. Python instructions resemble the English language, which is what makes it easier to learn and understand when compared to other programming languages.

**Python is:**

* a general purpose scripting language;
* a popular language used to code a variety of applications;
* a frequently used tool for automation;
* a cross-platform compatible language;
* a beginner-friendly language.

**Python is not:**

* a platform-specific / OS-specific scripting language;
* a client-side scripting language;
* a purely object-oriented programming language.

## Code comparison with Python

You will be learning about both Python and Bash scripting in this program. The following code illustrates a syntax difference between the two languages:

| **Print to screen in Python** | **Print to screen in Bash** |
| --- | --- |
| **>> print("Hello, how are you?")**  **Hello, how are you?** | **>> echo Hello, how are you?**  **Hello, how are you?** |

# Key Terms

* **Platform-specific / OS specific scripting language** - Platform-specific scripting languages, like PowerShell (for Windows) and Bash (for Linux), are used by system administrators on those platforms.
* **Client-side scripting language** - Client-side scripting languages, like JavaScript, are used mostly for web programming. The scripts are transferred from a web server to the end-user’s internet browser, then executed in the browser.
* **Machine language** - Machine language is the lowest-level computer language. It communicates directly with computing machines in binary code (ones and zeros). In binary code, one equals a pulse of electricity and zero equals no electrical pulse. Machine language instructions are made from translating languages like Python into complex patterns of ones and zeros.
* **Cross-platform** **language** - Programming language that is compatible with one or more platforms / operating systems (e.g., Windows, Linux, Mac, iOS, Android).
* **Object-oriented programming language** - In object-oriented programming languages, most coding elements are considered to be objects with configurable properties. For example, a form field is an object that can be configured to accept only dates as input in the mm/dd/yy format, and can be configured to read from and write to a specific database.
* **Python interpreter -** An interpreter is the program that reads and executes Python code by translating Python code into computer instructions.

# Resources

For additional Python practice, the following links will take you to several popular online interpreters and codepads:

* [Welcome to Python](https://www.python.org/shell/)
* [Online Python Interpreter](https://www.onlinegdb.com/online_python_interpreter)
* [Create a new Repl](https://repl.it/languages/python3)
* [Online Python-3 Compiler (Interpreter)](https://www.tutorialspoint.com/execute_python3_online.php)
* [Compile Python 3 Online](https://rextester.com/l/python3_online_compiler)
* [Your Python Trinket](https://trinket.io/python3)

# Study Guide: First Programming Concepts

This study guide provides a quick-reference summary of what you learned in this lesson and serves as a guide for the upcoming practice quiz.

# Functions

A function is a piece of code that performs a unit of work. In the examples you've seen so far, you have only encountered the **print()** function, which outputs a message to the screen. You will use this function frequently in this course to check the results of your code. The syntax of the print() function is modeled in the example below.

1

2

3

# Syntax for printing a string of text. Click Run to check the result.

print("Hello world!")





RunReset

# Keywords

A keyword is a reserved word in a programming language that performs a specific purpose. In your first Python example, you briefly encountered the keywords **for** and **in**. Note that keywords will often appear in **bold** in this course.

In the next few weeks, you will also learn the following keywords:

Values: **True**, **False**, **None** Conditions: **if**, **elif**, **else** Logical operators: **and**, **or**, **not** Loops: **for**, **in**, **while**, **break**, **continue** Functions: **def**, **return**

You don't need to learn this whole list now. We'll dive into each keyword as we encounter them. There are additional reserved keywords in Python. If you would like to read about them, please visit the linked “Python Keywords” article in the Resources section at the end of this study guide.

# Arithmetic operators

Python can calculate numbers using common mathematical operators, along with some special operators, too:

**x + y**            Addition + operator returns the sum of x plus y **x - y**             Subtraction - operator returns the difference of x minus y **x \* y**            Multiplication \* operator returns the product of x times y **x / y**             Division / operator returns the quotient of x divided by y **x\*\*e**            Exponent \*\* operator returns the result of raising x to the power of e  **x\*\*2**            Square expression returns x squared **x\*\*3**            Cube expression returns x cubed **x\*\*(1/2)**   Square root (½) or (0.5) fractional exponent operator returns the square root of x **x // y**           Floor division operator returns the integer part of the integer division of x by y **x % y**          Modulo operator returns the remainder part of the integer division of x by y

## Order of operations

The order of operations are to be calculated from left to right in the following order:

1. **P**arentheses ( ), { }, [ ]
2. **E**xponents xe   (x\*\*e)
3. **M**ultiplication \* and **D**ivision /
4. **A**ddition + and **S**ubtraction -

You might find the **PEMDAS** mnemonic device to be helpful in remembering the order.

# Resources for more information

For more information about the concepts covered in this reading, please visit:

* [Built-in Functions](https://docs.python.org/3/library/functions.html) - Lists and summarizes Python’s built-in functions.
* [Python Keywords](https://www.w3schools.com/python/python_ref_keywords.asp) - Lists Python’s reserved keywords and a brief description of what each keyword does.
* [Different Arithmetic operators in Python](https://flexiple.com/python/arithmetic-operators-in-python/) - Provides more examples of the proper syntax for using arithmetic operators in Python.

For additional Python practice, the following links will take you to several popular online interpreters and codepads:

* [Welcome to Python](https://www.python.org/shell/)
* [Online Python Interpreter](https://www.onlinegdb.com/online_python_interpreter)
* [Create a new Repl](https://repl.it/languages/python3)
* [Online Python-3 Compiler (Interpreter)](https://www.tutorialspoint.com/execute_python3_online.php)
* [Compile Python 3 Online](https://rextester.com/l/python3_online_compiler)
* [Your Python Trinket](https://trinket.io/python3)

# Study Guide: Week 1 Graded Quiz

# Study Guide: Week 1 Graded Assessment

It is time to prepare for your first graded quiz! Please review the following items from this module before beginning the Week 1 Graded Quiz. If you would like to refresh your memory on these materials, please revisit the Study Guides located before each Practice Quiz in Week 1: [Study Guide: Introduction to Programming](https://www.coursera.org/learn/python-crash-course/supplement/JNRad/study-guide-introduction-to-programming), [Study Guide: Introduction to Python](https://www.coursera.org/learn/python-crash-course/supplement/BqgFu/study-guide-introduction-to-python), and [Study Guide: First Programming Concepts](https://www.coursera.org/learn/python-crash-course/supplement/e5FGg/study-guide-first-programming-concepts).

# Knowledge

* Benefits of the Python programming language
* How Python compares to other programming languages
* How the knowledge of one programming language affects learning and using other programming languages
* How scripting applies to automation
* Proper syntax for arithmetic operations
* Functions and keywords used to display data
* Why precision is important when programming computer instructions

## Terms

* Computer programs
* Programming language
* Syntax
* Semantics
* Logic errors
* Script
* Automation
* Function

# Coding skills

## ****Skill 1****

* Use the print() function to output a string

1

2

3

4

5

6

7

8

9

10

11

12

13

# Syntax for printing a string of text

print("I love Python!")

# Syntax for printing numeric values

print(360)

print(32\*45)

# Syntax for printing the value of a variable

value = 8\*6

print(value)





RunReset

## ****Skill 2****

* Use arithmetic operators, with a focus on exponents

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

# Multiplication, division, addition, and subtraction

print(3\*8/2+5-1)

# Exponents

print(4\*\*6) # Syntax means 4 to the power of 6

print(4\*\*2) # To square a number

print(4\*\*3) # To cube a number

print(4\*\*0.5) # To find the square root of a number

# To calculate how many different possible combinations can be

# formed using a set of "x" characters with each character in "x"

# having "y" number of possible values, you will need to use an

# exponent for the calculation:

x = 4

y = 26

print(y\*\*x)





RunReset

## ****Skill 3****

* Use variables with assignment and arithmetic operators

1

2

3

4

5

6

7

# Assignment of values to the variables:

years = 10

weeks\_in\_a\_year = 52

# This variable is assigned an arithmetic calculation:

weeks\_in\_a\_decade = years \* weeks\_in\_a\_year

# Prints the calculation stored in the "weeks\_in\_a\_decade" variable:

print(weeks\_in\_a\_decade)





RunReset

# Reminder: Correct syntax is critical

Using precise syntax is critical when writing code in any programming language, including Python. Even a small typo can cause a syntax error and the automated Python-coded quiz grader will mark your code as incorrect. This reflects real life coding errors in the sense that a single error in spelling, case, punctuation, etc. can cause your code to fail. Coding problems caused by imprecise syntax will always be an issue whether you are learning a programming language or you are using programming skills on the job. So, it is critical to start the habit of being precise in your code now.

No credit will be given if there are any coding errors on the automated graded quizzes - including minor errors. Fortunately, you have 3 optional retake opportunities on the graded quizzes in this course. Additionally, you have unlimited retakes on practice quizzes and can review the videos and readings as many times as you need to master the concepts in this course.

Now, before starting the graded quiz, review this list of common syntax errors coders make when writing code.

### ****Common syntax errors:****

* Misspellings
* Incorrect indentations
* Missing or incorrect key characters:
  + Parenthetical types - ( curved ), [ square ], { curly }
  + Quote types - "straight-double" or 'straight-single', “curly-double” or ‘curly-single’
  + Block introduction characters, like colons - :
* Data type mismatches
* Missing, incorrectly used, or misplaced Python reserved words
* Using the wrong case (uppercase/lowercase) - Python is a case-sensitive language

# Resources

For additional Python practice, the following links will take you to several popular online interpreters and codepads:

* [Welcome to Python](https://www.python.org/shell/)
* [Online Python Interpreter](https://www.onlinegdb.com/online_python_interpreter)
* [Create a new Repl](https://repl.it/languages/python3)
* [Online Python-3 Compiler (Interpreter)](https://www.tutorialspoint.com/execute_python3_online.php)
* [Compile Python 3 Online](https://rextester.com/l/python3_online_compiler)
* [Your Python Trinket](https://trinket.io/python3)
* Built-in Data Types
* In programming, data type is an important concept.
* Variables can store data of different types, and different types can do different things.
* Python has the following data types built-in by default, in these categories:

|  |  |
| --- | --- |
| Text Type: | str |
| Numeric Types: | int, float, complex |
| Sequence Types: | list, tuple, range |
| Mapping Type: | dict |
| Set Types: | set, frozenset |
| Boolean Type: | bool |
| Binary Types: | bytes, bytearray, memoryview |

* Related Pages
* [Python Data Types Tutorial](https://www.w3schools.com/python/python_datatypes.asp)[Getting Data Type](https://www.w3schools.com/python/gloss_python_getting_data_type.asp)[Setting Data Type](https://www.w3schools.com/python/gloss_python_setting_data_type.asp)

|  |  |
| --- | --- |
| **Feature** | **Description** |
| [Indentation](https://www.w3schools.com/python/gloss_python_indentation.asp) | Indentation refers to the spaces at the beginning of a code line |
| [Comments](https://www.w3schools.com/python/gloss_python_comments.asp) | Comments are code lines that will not be executed |
| [Multiline Comments](https://www.w3schools.com/python/gloss_python_multi_line_comments.asp) | How to insert comments on multiple lines |
| [Creating Variables](https://www.w3schools.com/python/gloss_python_creating_variables.asp) | Variables are containers for storing data values |
| [Variable Names](https://www.w3schools.com/python/gloss_python_variable_names.asp) | How to name your variables |
| [Assign Values to Multiple Variables](https://www.w3schools.com/python/gloss_python_assign_value_to_multiple_variables.asp) | How to assign values to multiple variables |
| [Output Variables](https://www.w3schools.com/python/gloss_python_variable_output.asp) | Use the print statement to output variables |
| [String Concatenation](https://www.w3schools.com/python/gloss_python_string_concatenation.asp) | How to combine strings |
| [Global Variables](https://www.w3schools.com/python/gloss_python_global_variables.asp) | Global variables are variables that belongs to the global scope |
| [Built-In Data Types](https://www.w3schools.com/python/gloss_python_built-in_data_types.asp) | Python has a set of built-in data types |
| [Getting Data Type](https://www.w3schools.com/python/gloss_python_getting_data_type.asp) | How to get the data type of an object |
| [Setting Data Type](https://www.w3schools.com/python/gloss_python_setting_data_type.asp) | How to set the data type of an object |
| [Numbers](https://www.w3schools.com/python/gloss_python_numbers.asp) | There are three numeric types in Python |
| [Int](https://www.w3schools.com/python/gloss_python_int.asp) | The integer number type |
| [Float](https://www.w3schools.com/python/gloss_python_float.asp) | The floating number type |
| [Complex](https://www.w3schools.com/python/gloss_python_complex.asp) | The complex number type |
| [Type Conversion](https://www.w3schools.com/python/gloss_python_type_conversion.asp) | How to convert from one number type to another |
| [Random Number](https://www.w3schools.com/python/gloss_python_random_number.asp) | How to create a random number |
| [Specify a Variable Type](https://www.w3schools.com/python/gloss_python_specify_variable_type.asp) | How to specify a certain data type for a variable |
| [String Literals](https://www.w3schools.com/python/gloss_python_string_literals.asp) | How to create string literals |
| [Assigning a String to a Variable](https://www.w3schools.com/python/gloss_python_assign_string_variable.asp) | How to assign a string value to a variable |
| [Multiline Strings](https://www.w3schools.com/python/gloss_python_multi_line_strings.asp) | How to create a multiline string |
| [Strings are Arrays](https://www.w3schools.com/python/gloss_python_strings_are_arrays.asp) | Strings in Python are arrays of bytes representing Unicode characters |
| [Slicing a String](https://www.w3schools.com/python/gloss_python_string_slice.asp) | How to slice a string |
| [Negative Indexing on a String](https://www.w3schools.com/python/gloss_python_string_negative_indexing.asp) | How to use negative indexing when accessing a string |
| [String Length](https://www.w3schools.com/python/gloss_python_string_length.asp) | How to get the length of a string |
| [Check In String](https://www.w3schools.com/python/gloss_python_check_string.asp) | How to check if a string contains a specified phrase |
| [Format String](https://www.w3schools.com/python/gloss_python_string_format.asp) | How to combine two strings |
| [Escape Characters](https://www.w3schools.com/python/gloss_python_escape_characters.asp) | How to use escape characters |
| [Boolean Values](https://www.w3schools.com/python/gloss_python_boolean_values.asp) | True or False |
| [Evaluate Booleans](https://www.w3schools.com/python/gloss_python_evaluate_boolean_values.asp) | Evaluate a value or statement and return either True or False |
| [Return Boolean Value](https://www.w3schools.com/python/gloss_python_return_boolean.asp) | Functions that return a Boolean value |
| [Operators](https://www.w3schools.com/python/gloss_python_operators.asp) | Use operator to perform operations in Python |
| [Arithmetic Operators](https://www.w3schools.com/python/gloss_python_arithmetic_operators.asp) | Arithmetic operator are used to perform common mathematical operations |
| [Assignment Operators](https://www.w3schools.com/python/gloss_python_assignment_operators.asp) | Assignment operators are use to assign values to variables |
| [Comparison Operators](https://www.w3schools.com/python/gloss_python_comparison_operators.asp) | Comparison operators are used to compare two values |
| [Logical Operators](https://www.w3schools.com/python/gloss_python_logical_operators.asp) | Logical operators are used to combine conditional statements |
| [Identity Operators](https://www.w3schools.com/python/gloss_python_identity_operators.asp) | Identity operators are used to see if two objects are in fact the same object |
| [Membership Operators](https://www.w3schools.com/python/gloss_python_membership_operators.asp) | Membership operators are used to test is a sequence is present in an object |
| [Bitwise Operators](https://www.w3schools.com/python/gloss_python_bitwise_operators.asp) | Bitwise operators are used to compare (binary) numbers |
| [Lists](https://www.w3schools.com/python/gloss_python_lists.asp) | A list is an ordered, and changeable, collection |
| [Access List Items](https://www.w3schools.com/python/gloss_python_access_list_items.asp) | How to access items in a list |
| [Change List Item](https://www.w3schools.com/python/gloss_python_change_list_item.asp) | How to change the value of a list item |
| [Loop Through List Items](https://www.w3schools.com/python/gloss_python_loop_list_items.asp) | How to loop through the items in a list |
| [List Comprehension](https://www.w3schools.com/python/gloss_python_list_comprehension.asp) | How use a list comprehensive |
| [Check if List Item Exists](https://www.w3schools.com/python/gloss_python_check_if_list_item_exists.asp) | How to check if a specified item is present in a list |
| [List Length](https://www.w3schools.com/python/gloss_python_list_length.asp) | How to determine the length of a list |
| [Add List Items](https://www.w3schools.com/python/gloss_python_add_list_items.asp) | How to add items to a list |
| [Remove List Items](https://www.w3schools.com/python/gloss_python_remove_list_items.asp) | How to remove list items |
| [Copy a List](https://www.w3schools.com/python/gloss_python_copy_list.asp) | How to copy a list |
| [Join Two Lists](https://www.w3schools.com/python/gloss_python_join_lists.asp) | How to join two lists |
| [Tuple](https://www.w3schools.com/python/gloss_python_tuple.asp) | A tuple is an ordered, and unchangeable, collection |
| [Access Tuple Items](https://www.w3schools.com/python/gloss_python_access_tuple_items.asp) | How to access items in a tuple |
| [Change Tuple Item](https://www.w3schools.com/python/gloss_python_change_tuple_item.asp) | How to change the value of a tuple item |
| [Loop List Items](https://www.w3schools.com/python/gloss_python_loop_tuple_items.asp) | How to loop through the items in a tuple |
| [Check if Tuple Item Exists](https://www.w3schools.com/python/gloss_python_check_if_tuple_item_exists.asp) | How to check if a specified item is present in a tuple |
| [Tuple Length](https://www.w3schools.com/python/gloss_python_tuple_length.asp) | How to determine the length of a tuple |
| [Tuple With One Item](https://www.w3schools.com/python/gloss_python_tuple_one_item.asp) | How to create a tuple with only one item |
| [Remove Tuple Items](https://www.w3schools.com/python/gloss_python_remove_tuple_items.asp) | How to remove tuple items |
| [Join Two Tuples](https://www.w3schools.com/python/gloss_python_join_tuple.asp) | How to join two tuples |
| [Set](https://www.w3schools.com/python/gloss_python_set.asp) | A set is an unordered, and unchangeable, collection |
| [Access Set Items](https://www.w3schools.com/python/gloss_python_access_set_items.asp) | How to access items in a set |
| [Add Set Items](https://www.w3schools.com/python/gloss_python_add_set_items.asp) | How to add items to a set |
| [Loop Set Items](https://www.w3schools.com/python/gloss_python_loop_set_items.asp) | How to loop through the items in a set |
| [Check if Set Item Exists](https://www.w3schools.com/python/gloss_python_check_if_set_item_exists.asp) | How to check if a item exists |
| [Set Length](https://www.w3schools.com/python/gloss_python_set_length.asp) | How to determine the length of a set |
| [Remove Set Items](https://www.w3schools.com/python/gloss_python_remove_set_items.asp) | How to remove set items |
| [Join Two Sets](https://www.w3schools.com/python/gloss_python_join_sets.asp) | How to join two sets |
| [Dictionary](https://www.w3schools.com/python/gloss_python_dictionary.asp) | A dictionary is an unordered, and changeable, collection |
| [Access Dictionary Items](https://www.w3schools.com/python/gloss_python_access_dictionary_items.asp) | How to access items in a dictionary |
| [Change Dictionary Item](https://www.w3schools.com/python/gloss_python_change_dictionary_item.asp) | How to change the value of a dictionary item |
| [Loop Dictionary Items](https://www.w3schools.com/python/gloss_python_loop_dictionary_items.asp) | How to loop through the items in a tuple |
| [Check if Dictionary Item Exists](https://www.w3schools.com/python/gloss_python_check_if_dictionary_item_exists.asp) | How to check if a specified item is present in a dictionary |
| [Dictionary Length](https://www.w3schools.com/python/gloss_python_dictionary_length.asp) | How to determine the length of a dictionary |
| [Add Dictionary Item](https://www.w3schools.com/python/gloss_python_dictionary_add_item.asp) | How to add an item to a dictionary |
| [Remove Dictionary Items](https://www.w3schools.com/python/gloss_python_remove_dictionary_items.asp) | How to remove dictionary items |
| [Copy Dictionary](https://www.w3schools.com/python/gloss_python_copy_dictionary.asp) | How to copy a dictionary |
| [Nested Dictionaries](https://www.w3schools.com/python/gloss_python_nested_dictionaries.asp) | A dictionary within a dictionary |
| [If Statement](https://www.w3schools.com/python/gloss_python_if_statement.asp) | How to write an if statement |
| [If Indentation](https://www.w3schools.com/python/gloss_python_if_indentation.asp) | If statements in Python relies on indentation (whitespace at the beginning of a line) |
| [Elif](https://www.w3schools.com/python/gloss_python_elif.asp) | elif is the same as "else if" in other programming languages |
| [Else](https://www.w3schools.com/python/gloss_python_else.asp) | How to write an if...else statement |
| [Shorthand If](https://www.w3schools.com/python/gloss_python_if_shorthand.asp) | How to write an if statement in one line |
| [Shorthand If Else](https://www.w3schools.com/python/gloss_python_if_else_shorthand.asp) | How to write an if...else statement in one line |
| [If AND](https://www.w3schools.com/python/gloss_python_if_and.asp) | Use the and keyword to combine if statements |
| [If OR](https://www.w3schools.com/python/gloss_python_if_or.asp) | Use the or keyword to combine if statements |
| [If NOT](https://www.w3schools.com/python/gloss_python_if_not.asp) | Use the not keyword to reverse the condition |
| [Nested If](https://www.w3schools.com/python/gloss_python_if_nested.asp) | How to write an if statement inside an if statement |
| [The pass Keyword in If](https://www.w3schools.com/python/gloss_python_if_pass.asp) | Use the pass keyword inside empty if statements |
| [While](https://www.w3schools.com/python/gloss_python_while.asp) | How to write a while loop |
| [While Break](https://www.w3schools.com/python/gloss_python_while_break.asp) | How to break a while loop |
| [While Continue](https://www.w3schools.com/python/gloss_python_while_continue.asp) | How to stop the current iteration and continue wit the next |
| [While Else](https://www.w3schools.com/python/gloss_python_while_else.asp) | How to use an else statement in a while loop |
| [For](https://www.w3schools.com/python/gloss_python_for.asp) | How to write a for loop |
| [Loop Through a String](https://www.w3schools.com/python/gloss_python_for_string.asp) | How to loop through a string |
| [For Break](https://www.w3schools.com/python/gloss_python_for_break.asp) | How to break a for loop |
| [For Continue](https://www.w3schools.com/python/gloss_python_for_continue.asp) | How to stop the current iteration and continue wit the next |
| [Looping Through a range](https://www.w3schools.com/python/gloss_python_for_range.asp) | How to loop through a range of values |
| [For Else](https://www.w3schools.com/python/gloss_python_for_else.asp) | How to use an else statement in a for loop |
| [Nested Loops](https://www.w3schools.com/python/gloss_python_for_nested.asp) | How to write a loop inside a loop |
| [For pass](https://www.w3schools.com/python/gloss_python_for_pass.asp) | Use the pass keyword inside empty for loops |
| [Function](https://www.w3schools.com/python/gloss_python_function.asp) | How to create a function in Python |
| [Call a Function](https://www.w3schools.com/python/gloss_python_function_call.asp) | How to call a function in Python |
| [Function Arguments](https://www.w3schools.com/python/gloss_python_function_arguments.asp) | How to use arguments in a function |
| [\*args](https://www.w3schools.com/python/gloss_python_function_arbitrary_arguments.asp) | To deal with an unknown number of arguments in a function, use the \* symbol before the parameter name |
| [Keyword Arguments](https://www.w3schools.com/python/gloss_python_function_keyword_arguments.asp) | How to use keyword arguments in a function |
| [\*\*kwargs](https://www.w3schools.com/python/gloss_python_function_arbitrary_keyword_arguments.asp) | To deal with an unknown number of keyword arguments in a function, use the \* symbol before the parameter name |
| [Default Parameter Value](https://www.w3schools.com/python/gloss_python_function_default_parameter.asp) | How to use a default parameter value |
| [Passing a List as an Argument](https://www.w3schools.com/python/gloss_python_function_passing_list.asp) | How to pass a list as an argument |
| [Function Return Value](https://www.w3schools.com/python/gloss_python_function_return_value.asp) | How to return a value from a function |
| [The pass Statement in Functions](https://www.w3schools.com/python/gloss_python_function_pass.asp) | Use the pass statement in empty functions |
| [Function Recursion](https://www.w3schools.com/python/gloss_python_function_recursion.asp) | Functions that can call itself is called recursive functions |
| [Lambda Function](https://www.w3schools.com/python/gloss_python_lambda.asp) | How to create anonymous functions in Python |
| [Why Use Lambda Functions](https://www.w3schools.com/python/gloss_python_lambda_why.asp) | Learn when to use a lambda function or not |
| [Array](https://www.w3schools.com/python/gloss_python_array.asp) | Lists can be used as Arrays |
| [What is an Array](https://www.w3schools.com/python/gloss_python_arrray_what_is.asp) | Arrays are variables that can hold more than one value |
| [Access Arrays](https://www.w3schools.com/python/gloss_python_array_access.asp) | How to access array items |
| [Array Length](https://www.w3schools.com/python/gloss_python_array_length.asp) | How to get the length of an array |
| [Looping Array Elements](https://www.w3schools.com/python/gloss_python_array_loop.asp) | How to loop through array elements |
| [Add Array Element](https://www.w3schools.com/python/gloss_python_array_add.asp) | How to add elements from an array |
| [Remove Array Element](https://www.w3schools.com/python/gloss_python_array_remove.asp) | How to remove elements from an array |
| [Array Methods](https://www.w3schools.com/python/gloss_python_array_methods.asp) | Python has a set of Array/Lists methods |
| [Class](https://www.w3schools.com/python/gloss_python_class.asp) | A class is like an object constructor |
| [Create Class](https://www.w3schools.com/python/gloss_python_class_create.asp) | How to create a class |
| [The Class \_\_init\_\_() Function](https://www.w3schools.com/python/gloss_python_class_init.asp) | The \_\_init\_\_() function is executed when the class is initiated |
| [Object Methods](https://www.w3schools.com/python/gloss_python_object_methods.asp) | Methods in objects are functions that belongs to the object |
| [self](https://www.w3schools.com/python/gloss_python_self.asp) | The self parameter refers to the current instance of the class |
| [Modify Object Properties](https://www.w3schools.com/python/gloss_python_object_modify_properties.asp) | How to modify properties of an object |
| [Delete Object Properties](https://www.w3schools.com/python/gloss_python_object_delete_properties.asp) | How to modify properties of an object |
| [Delete Object](https://www.w3schools.com/python/gloss_python_object_delete.asp) | How to delete an object |
| [Class pass Statement](https://www.w3schools.com/python/gloss_python_class_pass.asp) | Use the pass statement in empty classes |
| [Create Parent Class](https://www.w3schools.com/python/gloss_python_parent_class.asp) | How to create a parent class |
| [Create Child Class](https://www.w3schools.com/python/gloss_python_child_class.asp) | How to create a child class |
| [Create the \_\_init\_\_() Function](https://www.w3schools.com/python/gloss_python_create_init.asp) | How to create the \_\_init\_\_() function |
| [super Function](https://www.w3schools.com/python/gloss_python_super.asp) | The super() function make the child class inherit the parent class |
| [Add Class Properties](https://www.w3schools.com/python/gloss_python_class_add_properties.asp) | How to add a property to a class |
| [Add Class Methods](https://www.w3schools.com/python/gloss_python_class_add_methods.asp) | How to add a method to a class |
| [Iterators](https://www.w3schools.com/python/gloss_python_iterators.asp) | An iterator is an object that contains a countable number of values |
| [Iterator vs Iterable](https://www.w3schools.com/python/gloss_python_iterator_vs_iterable.asp) | What is the difference between an iterator and an iterable |
| [Loop Through an Iterator](https://www.w3schools.com/python/gloss_python_iterator_loop.asp) | How to loop through the elements of an iterator |
| [Create an Iterator](https://www.w3schools.com/python/gloss_python_iterator_create.asp) | How to create an iterator |
| [StopIteration](https://www.w3schools.com/python/gloss_python_iterator_stop.asp) | How to stop an iterator |
| [Global Scope](https://www.w3schools.com/python/gloss_python_global_scope.asp) | When does a variable belong to the global scope? |
| [Global Keyword](https://www.w3schools.com/python/gloss_python_global_keyword.asp) | The global keyword makes the variable global |
| [Create a Module](https://www.w3schools.com/python/gloss_python_module_create.asp) | How to create a module |
| [Variables in Modules](https://www.w3schools.com/python/gloss_python_module_variables.asp) | How to use variables in a module |
| [Renaming a Module](https://www.w3schools.com/python/gloss_python_module_rename.asp) | How to rename a module |
| [Built-in Modules](https://www.w3schools.com/python/gloss_python_module_built-in.asp) | How to import built-in modules |
| [Using the dir() Function](https://www.w3schools.com/python/gloss_python_module_dir.asp) | List all variable names and function names in a module |
| [Import From Module](https://www.w3schools.com/python/gloss_python_module_import_from.asp) | How to import only parts from a module |
| [Datetime Module](https://www.w3schools.com/python/gloss_python_date.asp) | How to work with dates in Python |
| [Date Output](https://www.w3schools.com/python/gloss_python_date_output.asp) | How to output a date |
| [Create a Date Object](https://www.w3schools.com/python/gloss_python_date_create.asp) | How to create a date object |
| [The strftime Method](https://www.w3schools.com/python/gloss_python_date_strftime.asp) | How to format a date object into a readable string |
| [Date Format Codes](https://www.w3schools.com/python/gloss_python_date_format_codes.asp) | The datetime module has a set of legal format codes |
| [JSON](https://www.w3schools.com/python/gloss_python_json.asp) | How to work with JSON in Python |
| [Parse JSON](https://www.w3schools.com/python/gloss_python_json_parse.asp) | How to parse JSON code in Python |
| [Convert into JSON](https://www.w3schools.com/python/gloss_python_convert_into_JSON.asp) | How to convert a Python object in to JSON |
| [Format JSON](https://www.w3schools.com/python/gloss_python_format_json.asp) | How to format JSON output with indentations and line breaks |
| [Sort JSON](https://www.w3schools.com/python/gloss_python_json_sort.asp) | How to sort JSON |
| [RegEx Module](https://www.w3schools.com/python/gloss_python_regex.asp) | How to import the regex module |
| [RegEx Functions](https://www.w3schools.com/python/gloss_python_regex_functions.asp) | The re module has a set of functions |
| [Metacharacters in RegEx](https://www.w3schools.com/python/gloss_python_regex_metacharacters.asp) | Metacharacters are characters with a special meaning |
| [RegEx Special Sequences](https://www.w3schools.com/python/gloss_python_regex_sequences.asp) | A backslash followed by a a character has a special meaning |
| [RegEx Sets](https://www.w3schools.com/python/gloss_python_regex_sets.asp) | A set is a set of characters inside a pair of square brackets with a special meaning |
| [RegEx Match Object](https://www.w3schools.com/python/gloss_python_regex_match.asp) | The Match Object is an object containing information about the search and the result |
| [Install PIP](https://www.w3schools.com/python/gloss_python_pip_install.asp) | How to install PIP |
| [PIP Packages](https://www.w3schools.com/python/gloss_python_pip_packages.asp) | How to download and install a package with PIP |
| [PIP Remove Package](https://www.w3schools.com/python/gloss_python_pip_packages_remove.asp) | How to remove a package with PIP |
| [Error Handling](https://www.w3schools.com/python/gloss_python_error_handling.asp) | How to handle errors in Python |
| [Handle Many Exceptions](https://www.w3schools.com/python/gloss_python_try_except.asp) | How to handle more than one exception |
| [Try Else](https://www.w3schools.com/python/gloss_python_try_else.asp) | How to use the else keyword in a try statement |
| [Try Finally](https://www.w3schools.com/python/gloss_python_try_finally.asp) | How to use the finally keyword in a try statement |
| [raise](https://www.w3schools.com/python/gloss_python_raise.asp) | How to raise an exception in Python |

# **Python Random Module**

[❮ Previous](https://www.w3schools.com/python/python_ref_glossary.asp)[Next ❯](https://www.w3schools.com/python/module_requests.asp)

Python has a built-in module that you can use to make random numbers.

The random module has a set of methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| [seed()](https://www.w3schools.com/python/ref_random_seed.asp) | Initialize the random number generator |
| [getstate()](https://www.w3schools.com/python/ref_random_getstate.asp) | Returns the current internal state of the random number generator |
| [setstate()](https://www.w3schools.com/python/ref_random_setstate.asp) | Restores the internal state of the random number generator |
| [getrandbits()](https://www.w3schools.com/python/ref_random_getrandbits.asp) | Returns a number representing the random bits |
| [randrange()](https://www.w3schools.com/python/ref_random_randrange.asp) | Returns a random number between the given range |
| [randint()](https://www.w3schools.com/python/ref_random_randint.asp) | Returns a random number between the given range |
| [choice()](https://www.w3schools.com/python/ref_random_choice.asp) | Returns a random element from the given sequence |
| [choices()](https://www.w3schools.com/python/ref_random_choices.asp) | Returns a list with a random selection from the given sequence |
| [shuffle()](https://www.w3schools.com/python/ref_random_shuffle.asp) | Takes a sequence and returns the sequence in a random order |
| [sample()](https://www.w3schools.com/python/ref_random_sample.asp) | Returns a given sample of a sequence |
| [random()](https://www.w3schools.com/python/ref_random_random.asp) | Returns a random float number between 0 and 1 |
| [uniform()](https://www.w3schools.com/python/ref_random_uniform.asp) | Returns a random float number between two given parameters |
| [triangular()](https://www.w3schools.com/python/ref_random_triangular.asp) | Returns a random float number between two given parameters, you can also set a mode parameter to specify the midpoint between the two other parameters |
| betavariate() | Returns a random float number between 0 and 1 based on the Beta distribution (used in statistics) |
| expovariate() | Returns a random float number based on the Exponential distribution (used in statistics) |
| gammavariate() | Returns a random float number based on the Gamma distribution (used in statistics) |
| gauss() | Returns a random float number based on the Gaussian distribution (used in probability theories) |
| lognormvariate() | Returns a random float number based on a log-normal distribution (used in probability theories) |
| normalvariate() | Returns a random float number based on the normal distribution (used in probability theories) |
| vonmisesvariate() | Returns a random float number based on the von Mises distribution (used in directional statistics) |
| paretovariate() | Returns a random float number based on the Pareto distribution (used in probability theories) |
| weibullvariate() | Returns a random float number based on the Weibull distribution (used in statistics) |

# **Python Requests Module**

[❮ Previous](https://www.w3schools.com/python/module_random.asp)[Next ❯](https://www.w3schools.com/python/module_statistics.asp)

### Example[Get your own Python Server](https://www.w3schools.com/spaces/)

Make a request to a web page, and print the response text:

import requests  
  
x = requests.get('https://w3schools.com/python/demopage.htm')  
  
print(x.text)

[Run Example »](https://www.w3schools.com/python/showpython.asp?filename=demo_requests_get_url)

## Definition and Usage

The requests module allows you to send HTTP requests using Python.

The HTTP request returns a [Response Object](https://www.w3schools.com/python/ref_requests_response.asp) with all the response data (content, encoding, status, etc).

## Download and Install the Requests Module

Navigate your command line to the location of PIP, and type the following:

C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip install requests

## Syntax

requests.methodname(params)

## Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| [delete(url, args)](https://www.w3schools.com/python/ref_requests_delete.asp) | Sends a DELETE request to the specified url |
| [get(url, params, args)](https://www.w3schools.com/python/ref_requests_get.asp) | Sends a GET request to the specified url |
| [head(url, args)](https://www.w3schools.com/python/ref_requests_head.asp) | Sends a HEAD request to the specified url |
| patch(url, data, args) | Sends a PATCH request to the specified url |
| [post(url, data, json, args)](https://www.w3schools.com/python/ref_requests_post.asp) | Sends a POST request to the specified url |
| put(url, data, args) | Sends a PUT request to the specified url |
| request(method, url, args) | Sends a request of the specified method to the specified url |

# Implicit vs Explicit Conversion

As we saw earlier in the video, some data types can be mixed and matched due to implicit conversion. Implicit conversion is where the interpreter helps us out and automatically converts one data type into another, without having to explicitly tell it to do so.

By contrast, explicit conversion is where we manually convert from one data type to another by calling the relevant function for the data type we want to convert to. We used this in our video example when we wanted to print a number alongside some text. Before we could do that, we needed to call the str() function to convert the number into a string. Once the number was explicitly converted to a string, we could join it with the rest of our textual string and print the result.

# Defining Functions Recap

We looked at a few examples of built-in functions in Python, but being able to define your own functions is incredibly powerful. We start a function definition with the def keyword, followed by the name we want to give our function. After the name, we have the parameters, also called arguments, for the function enclosed in parentheses. A function can have no parameters, or it can have multiple parameters. Parameters allow us to call a function and pass it data, with the data being available inside the function as variables with the same name as the parameters. Lastly, we put a colon at the end of the line.

After the colon, the function body starts. It’s important to note that in Python the function body is delimited by indentation. This means that all code indented to the right following a function definition is part of the function body. The first line that’s no longer indented is the boundary of the function body. It’s up to you how many spaces you use when indenting -- just make sure to be consistent. So if you choose to indent with four spaces, you need to use four spaces everywhere in your code.

# Returning Values Using Functions

Sometimes we don't want a function to simply run and finish. We may want a function to manipulate data we passed it and then return the result to us. This is where the concept of return values comes in handy. We use the return keyword in a function, which tells the function to pass data back. When we call the function, we can store the returned value in a variable. Return values allow our functions to be more flexible and powerful, so they can be reused and called multiple times.

Functions can even return multiple values. Just don't forget to store all returned values in variables! You could also have a function return nothing, in which case the function simply exits.

# Study Guide: Functions

This study guide provides a quick-reference summary of what you learned in this lesson and serves as a guide for the upcoming practice quiz.

In the Functions segment, you learned how to define and call functions, utilize a function’s parameters, and return data from a function. You also learned how to differentiate and convert between different data types utilizing variables. Plus, you learned a few best practices for writing reusable and readable code.

# Terms

* **return value** - the value or variable returned as the end result of a function
* **parameter (argument)** -  a value passed into a function for use within the function
* **refactoring code** - a process to restructure code without changing functionality

# Knowledge

* The purpose of the **def()** keywordis to define a new function.
* Best practices for writing code that is readable and reusable:
  + **Create a reusable function** - Replace duplicate code with one reusable function to make the code easier to read and repurpose.
  + **Refactor code** - Update code so that it is self-documenting and the intent of the code is clear.
  + **Add comments** - Adding comments is part of creating self-documenting code. Using comments allows you to leave notes to yourself and/or other programmers to make the purpose of the code clear.

# Coding skills

**Skill Group 1**

* Use a function that accepts multiple parameters
* Return a result value

# Coding skills

**Skill Group 1**

* Use a function that accepts multiple parameters
* Return a result value

1

2

3

4

5

6

7

8

9

10

11

12

13

14

# This function calculates the number of days in a variable number of

# years, months, and days. These variables are provided by the user and

# are passed to the function through the function’s parameters.

def find\_total\_days(years, months, days):

# Assign a variable to hold the calculations for the number of days in

# a year (years\*365) plus the number of days in a month (months\*30) plus

# the number of days provided through the "days" parameter variable.

    my\_days = (years\*365) + (months\*30) + days

# Use the "return" keyword to send the result of the "my\_days"

# calculation to the function call.

    return my\_days

# Function call with user provided parameter values.

print(find\_total\_days(2,5,23))





RunReset

**Skill Group 2**

* Use a function to return the result of a measurement conversion
* Use arithmetic operators to perform a calculation
* Combine text with a function call within a print() statement
* Convert the return value from a float data type to a string for the print() function
* Call the function and perform a calculation on the return value within a print() statement

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

# This function converts fluid ounces to milliliters and returns the

# result of the conversion.

def convert\_volume(fluid\_ounce):

# Calculate value of the "ml" variable using the parameter variable

# "fluid\_ounce". There are approximately 29.5 milliliters in 1 fluid

# ounce.

    ml = fluid\_ounce \* 29.5

# Return the result of the calculation.

    return ml

# Call the conversion from within the print() function using 2 fluid

# ounces. Convert the return value from a float to a string.

print("The volume in millimeters is " + str(convert\_volume(2)))

# Call the function again and double the 2 fluid ounces from within

# the print function.

print("The volume in millimeters is " + str(convert\_volume(2)\*2))

# Alternative calculation:

# print("The volume in millimeters is " + str(convert\_volume(4))





RunReset

# Python practice information

For additional Python practice, the following links will take you to several popular online interpreters and codepads:

* [Welcome to Python](https://www.python.org/shell/)
* [Online Python Interpreter](https://www.onlinegdb.com/online_python_interpreter)
* [Create a new Repl](https://repl.it/languages/python3)
* [Online Python-3 Compiler (Interpreter)](https://www.tutorialspoint.com/execute_python3_online.php)
* [Compile Python 3 Online](https://rextester.com/l/python3_online_compiler)
* [Your Python Trinket](https://trinket.io/python3)

# Study Guide: Functions

This study guide provides a quick-reference summary of what you learned in this lesson and serves as a guide for the upcoming practice quiz.

In the Functions segment, you learned how to define and call functions, utilize a function’s parameters, and return data from a function. You also learned how to differentiate and convert between different data types utilizing variables. Plus, you learned a few best practices for writing reusable and readable code.

# Terms

* **return value** - the value or variable returned as the end result of a function
* **parameter (argument)** -  a value passed into a function for use within the function
* **refactoring code** - a process to restructure code without changing functionality

# Knowledge

* The purpose of the **def()** keywordis to define a new function.
* Best practices for writing code that is readable and reusable:
  + **Create a reusable function** - Replace duplicate code with one reusable function to make the code easier to read and repurpose.
  + **Refactor code** - Update code so that it is self-documenting and the intent of the code is clear.
  + **Add comments** - Adding comments is part of creating self-documenting code. Using comments allows you to leave notes to yourself and/or other programmers to make the purpose of the code clear.

# Coding skills

**Skill Group 1**

* Use a function that accepts multiple parameters
* Return a result value

1

2

3

4

5

6

7

8

9

10

11

12

13

14

# This function calculates the number of days in a variable number of

# years, months, and days. These variables are provided by the user and

# are passed to the function through the function’s parameters.

def find\_total\_days(years, months, days):

# Assign a variable to hold the calculations for the number of days in

# a year (years\*365) plus the number of days in a month (months\*30) plus

# the number of days provided through the "days" parameter variable.

    my\_days = (years\*365) + (months\*30) + days

# Use the "return" keyword to send the result of the "my\_days"

# calculation to the function call.

    return my\_days

# Function call with user provided parameter values.

print(find\_total\_days(2,5,23))





RunReset

**Skill Group 2**

* Use a function to return the result of a measurement conversion
* Use arithmetic operators to perform a calculation
* Combine text with a function call within a print() statement
* Convert the return value from a float data type to a string for the print() function
* Call the function and perform a calculation on the return value within a print() statement

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

# This function converts fluid ounces to milliliters and returns the

# result of the conversion.

def convert\_volume(fluid\_ounce):

# Calculate value of the "ml" variable using the parameter variable

# "fluid\_ounce". There are approximately 29.5 milliliters in 1 fluid

# ounce.

    ml = fluid\_ounce \* 29.5

# Return the result of the calculation.

    return ml

# Call the conversion from within the print() function using 2 fluid

# ounces. Convert the return value from a float to a string.

print("The volume in millimeters is " + str(convert\_volume(2)))

# Call the function again and double the 2 fluid ounces from within

# the print function.

print("The volume in millimeters is " + str(convert\_volume(2)\*2))

# Alternative calculation:

# print("The volume in millimeters is " + str(convert\_volume(4))





RunReset

# Python practice information

For additional Python practice, the following links will take you to several popular online interpreters and codepads:

* [Welcome to Python](https://www.python.org/shell/)
* [Online Python Interpreter](https://www.onlinegdb.com/online_python_interpreter)
* [Create a new Repl](https://repl.it/languages/python3)
* [Online Python-3 Compiler (Interpreter)](https://www.tutorialspoint.com/execute_python3_online.php)
* [Compile Python 3 Online](https://rextester.com/l/python3_online_compiler)
* [Your Python Trinket](https://trinket.io/python3)