- Map() to apply function to a list
- Zip() to convert 2+ lists to a tuple
- x

-
- But the process always remains the same—identifying a problem, [searching for if a similar problem has a current solution], coming up with a solution, determining the tools that help you achieve your solution, as well as the most significant part—writing the actual script
- Common Sequence Operations - Official python.org documentation for list, tuple, and range sequence operations.
- Lists - Official python.org documentation for list operations and methods.
- Tuples - Official python.org documentation for tuple operations and methods
- The official Python tutorial
- The Think Python
- The official language reference
- Read the official Python documentation.
- Search for answers or ask a question on Stack Overflow.
- Built-in Functions - Lists and summarizes Python's built-in functions.
- Python Keywords - Lists Python's reserved keywords and a brief description of what each keyword does.
- Different Arithmetic operators in Python - Provides more examples of the proper syntax for using arithmetic operators in Python.
- e PEP 8 Style Guide for Python so
- **Data Types**
- https://docs.python.org/3/library/functions.html#len
- Python - map() function
- Python zip() method
- String methods
- Formatting expressions: all available expressions.
- https://www.w3schools.com/python/python_dictionaries.asp
- https://www.coursera.org/learn/python-crash-course/supplement/Gz86W/python-in-action -import csv
- https://www.coursera.org/learn/python-crash-course/supplement/z9vVO/study-guide-module-4-graded-quiz

- GitHub has great resources on how to create projects here and how to store code here
- This article has tons of information on optimizing your LinkedIn profile. There is even a section focused on job transitions!
- This resource has lots of information on how to use GitHub as a job seeker.
-

Career

- November 2023, there were ~818K job openings in 2023 asking for Python skills.
- You can use Python to calculate statistics, run your e-commerce site, process images, interact with web services

https://www.coursera.org/learn/python-crash-course/supplement/PBgYL/finding-your-path-and-perfect-role

A generalist is knowledgeable about many topics and has various interests, while a specialist is an expert in a specific field.

Common Generalist Roles

- IT Support Specialist II
- IT Consultant
- IT Manager

Common Specialist Roles

- Automation Engineer
- Python Developer
- Software Engineer
- Cloud Engineer
-

Formatting expressions

Expr	Meaning	Example
{:d}	integer value	"{0:.0f}".format(10.5) → '10'
{:.2f}	floating point with that many decimals	'{:.2f}'.format(0.5) → '0.50'
{:.2s}	string with that many characters	'{:.2s}'.format('Python') → 'Py'
{:<6s}	string aligned to the left that many spaces	'{:<6s}'.format('Py') → 'Py '
{:>6s}	string aligned to the right that many spaces	'{:>6s}'.format('Py') → ' Py'
{:^6s}	string centered in that many spaces	'{:^6s}'.format('Py') → ' Py '

-

Study Guides

- https://www.coursera.org/learn/python-crash-course/supplement/ydylo/study-guide-strings#

- https://www.coursera.org/learn/python-crash-course/supplement/05vqZ/string-reference-guide
- https://www.coursera.org/learn/python-crash-course/supplement/l5DRc/study-guide-module-3-graded-quiz#
- https://www.coursera.org/learn/python-crash-course/supplement/iyNWi/study-guide-while-loops#
- https://www.coursera.org/learn/python-crash-course/supplement/dqH6E/study-guide-for-loops#
- https://www.coursera.org/learn/python-crash-course/supplement/L725I/study-guide-module-2-graded-quiz#
- https://www.coursera.org/learn/python-crash-course/supplement/SshSU/study-guide-functions#
- https://www.coursera.org/learn/python-crash-course/supplement/5Y1CI/study-guide-conditionals#
- https://www.coursera.org/learn/python-crash-course/supplement/JNRad/study-guide-introduction-to-programming
- https://www.coursera.org/learn/python-crash-course/supplement/BqgFu/study-guide-introduction-to-python
- https://www.coursera.org/learn/python-crash-course/supplement/e5FGg/study-guide-first-programming-concepts
- https://www.coursera.org/learn/python-crash-course/supplement/sbRdF/study-guide-list-operations-and-methods
-

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

- **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.

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

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

Common semantic errors:

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

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.

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.

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
xy**	Exponent ** operator returns the result of raising x to the power of y
x2**	Square expression returns x squared
x3**	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

Automation: The process of replacing a manual step with one that happens automatically
Client-side scripting language: Primarily for web programming; the scripts are transferred from a web server to the end-user's internet browser, then executed in the browser
Code editors: Tools to provide features, including syntax highlighting, automatic indentation, error checking, and autocompletion
Computer program: A step-by-step list of instructions that a computer follows to reach an intended goal
Functions: A reusable block of code that performs a specific task
IDE: A software application that provides comprehensive facilities for software development
Interpreter: The program that reads and executes code
Input: Information that is provided to a program by the end user
Logic errors: Errors in code that prevent it from running correctly

Machine language: Lowest-level computer language. It communicates directly with computing machines in binary code (ones and zeros)
Object-oriented programming language: Most coding elements are considered to be objects with configurable properties
Output: the end result of a task performed by a function or computer program
Platform-specific scripting language: Language used by system administrators on those specific platforms
Programming: The process of writing a program to behave in different ways
Programming code: A set of written computer instructions, guided by rules, using a computer programming language
Programming languages: Language with syntax and semantics to write computer programs
Python: A general purpose programming language
Python interpreter: Program that reads and executes Python code by translating Python code into computer instructions
Script: Often used to automate specific tasks
Semantics: The intended meaning or effect of statements, or collections of words, in both human and computer languages
Syntax: The rules for how each statements are constructed in both human and computer languages
Variables: These are used to temporarily store changeable values in programming 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

Keywords

```
in
not
or
for
while
return
+
-
*
```

/
**
%
//
>
<
==
Def
if
Elif
Else

Naming rules and conventions

When assigning names to objects, programmers adhere to a set of rules and conventions which help to standardize code and make it more accessible to everyone. Here are some naming rules and conventions that you should know:

- Names cannot contain spaces.
- Names may be a mixture of upper and lower case characters.
- Names can’t start with a number but may contain numbers after the first character.
- Variable names and function names should be written in snake_case, which means that all letters are lowercase and words are separated using an underscore.
- Descriptive names are better than cryptic abbreviations because they help other programmers (and you) read and interpret your code. For example, student_name is better than sn. It may feel excessive when you write it, but when you return to your code you’ll find it much easier to understand.

The Zen of Python

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one—and preferably only one—obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

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

Uppercase		Uppercase		Lowercase		Lowercase	
Unicode #	Character	Unicode #	Character	Unicode #	Character	Unicode #	Character
65	A	78	N	97	a	110	n
66	B	79	O	98	b	111	o
67	C	80	P	99	c	112	p
68	D	81	Q	100	d	113	q
69	E	82	R	101	e	114	r
70	F	83	S	102	f	115	s
71	G	84	T	103	g	116	t
72	H	85	U	104	h	117	u
73	I	86	V	105	i	118	v
74	J	87	W	106	j	119	w
75	K	88	X	107	k	120	x
76	L	89	Y	108	l	121	y
77	M	90	Z	109	m	122	z

Disadvantages of Python and Mitigation

- Python's dynamic typing can lead to errors that are only caught during runtime, potentially late in development.
- This drawback can be mitigated by writing thorough tests to ensure code reliability.

Terms and definitions from Course 1, Module 2

Built-in functions: Functions that exist within Python and can be called directly
Comments: Notes to yourself and/or other programmers to make the purpose of the code clear
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)
Explicit conversion: This occurs when code is written to manually convert one data type to another using a data type conversion function
Expression: A combination of numbers, symbols, or other values that produce a result when evaluated
Implicit conversion: This occurs when the Python interpreter automatically converts one data type to another
Logical operators: Operators used to combine or manipulate boolean values (True or False) to create complex conditions for decision-making.
Parameter (argument): A value passed into a function for use within the function, controlling the behavior of the CSV reader and writer
Refactoring: When a code is updated to be more self-documenting and clarify the intent
Return value: This is the value or variable returned as the end result of a function

Common errors in Loops

If you get an error message on a loop or it appears to hang, your debugging checklist should include the following checks:

- **Failure to initialize variables**. Make sure all the variables used in the loop's condition are initialized before the loop.
- **Unintended infinite loops**. Make sure that the body of the loop modifies the variables used in the condition, so that the loop will eventually end for all possible values of the variables. You can often prevent an infinite loop by using the break keyword or by adding end criteria to the condition part of the `while` loop.

Terms

- `while loop` - Tells the computer to execute a set of instructions while a specified condition is `True`. In other words, `while` loops keep executing the same group of instructions until the condition becomes `False`.
- `infinite loop` - Missing a method for exiting the loop, causing the loop to run forever.
- `break` - A `break` statement in Python provides a way to exit out of a loop before the loop's condition is false. Once a break statement is encountered, the program's control flow jumps out of the loop and continues executing the code after the loop.
- `pass` - A `pass` statement in Python is a placeholder statement which is used when the syntax requires a statement, but you don't want to execute any code or command.

Math concepts on the practice quiz

The coding problems on the upcoming practice quiz will involve a few math concepts. Don't worry if you are rusty on math. You will have plenty of support with these concepts on the quiz. The following is a quick overview of the math terms you will encounter on the quiz:

- **prime numbers** - Integers that have only two factors, which are the number itself multiplied by 1. The first ten prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, 23 and 29. Each of these prime numbers can be evenly divided only by itself and 1.
- **prime factors** - Prime numbers that are factors of an integer. For example, the prime numbers 2 and 5 are the prime factors of the number 10 (2x5=10). The prime factors of an integer will not produce a remainder when used to divide that integer.
- **divisor** - A number (denominator) that is used to divide another number (numerator). For example, if the number 10 is divided by 5, the number 5 is the divisor.
- **sum of all divisors of a number** - The result of adding all of the divisors of a number together.
- **multiplication table** - An integer multiplied by a series of numbers and their results formatted as a table or a list. For example:

You can use `for` loops with strings to perform tasks and functions such as:

- Reading text from a file
- Searching for a value or specific data in a document or spreadsheet

List-specific operations and methods

One major difference between lists and tuples is that lists are mutable (changeable) and tuples are immutable (not changeable). There are a few operations and methods that are specific to changing data within lists:

- `list[index] = x` - Replaces the element at index [n] with x.
- `list.append(x)` - Appends x to the end of the list.
- `list.insert(index, x)` - Inserts x at index position [index].
- `list.pop(index)` - Returns the element at [index] and removes it from the list. If [index] position is not in the list, the last element in the list is returned and removed.
- `list.remove(x)` - Removes the first occurrence of x in the list.
- `list.sort()` - Sorts the items in the list.
- `list.reverse()` - Reverses the order of items of the list.
- `list.clear()` - Deletes all items in the list.
- `list.copy()` - Creates a copy of the list.
- `list.extend(other_list)` - Appends all the elements of other_list at the end of list
- `map(function, iterable)` - Applies a given function to each item of an iterable (such as a list) and returns a map object with the results
- `zip(*iterables)` - Takes in iterables as arguments and returns an iterator that generates tuples, where the i-th tuple contains the i-th element from each of the argument iterables.

Tuple use cases

Remember, there are a number of cases where using a tuple might be more suitable than other data types:

- Protecting data: Because tuples are immutable, they can be used in situations where you want to ensure the data you have cannot be changed. This can be particularly helpful when dealing with sensitive or important information that should remain constant throughout the execution of a program.
- Hashable keys: Because they're immutable, tuples can be used as keys on dictionaries, which can be useful for complex keys.
- Efficiency: Tuples are generally more memory-efficient than lists, making them advantageous when dealing with large datasets.

Dictionaries

Operations

- **len(dictionary)** - Returns the number of items in a dictionary.
- **for key, in dictionary** - Iterates over each key in a dictionary.
- **for key, value in dictionary.items()** - Iterates over each key,value pair in a dictionary.
- **if key in dictionary** - Checks whether a key is in a dictionary.
- **dictionary[key]** - Accesses a value using the associated key from a dictionary.
- **dictionary[key] = value** - Sets a value associated with a key.
- **del dictionary[key]** - Removes a value using the associated key from a dictionary.

Methods

- **dictionary.get(key, default)** - Returns the value corresponding to a key, or the default value if the specified key is not present.
- **dictionary.keys()** - Returns a sequence containing the keys in a dictionary.
- **dictionary.values()** - Returns a sequence containing the values in a dictionary.
- **dictionary[key].append(value)** - Appends a new value for an existing key.
- **dictionary.update(other_dictionary)** - Updates a dictionary with the items from another dictionary. Existing entries are updated; new entries are added.
- **dictionary.clear()** - Deletes all items from a dictionary.
- **dictionary.copy()** - Makes a copy of a dictionary.

Dictionaries versus Lists

Dictionaries are similar to lists, but there are a few differences:

Both dictionaries and lists:

- are used to organize elements into collections;
- are used to initialize a new dictionary or list, use empty brackets;
- can iterate through the items or elements in the collection; and

- can use a variety of methods and operations to create and change the collections, like removing and inserting items or elements.

Dictionaries only:

- are unordered sets;
- have keys that can be a variety of data types, including strings, integers, floats, tuples;.
- can access dictionary values by keys;
- use square brackets inside curly brackets { [ ] };
- use colons between the key and the value(s);
- use commas to separate each key group and each value within a key group;
- make it quicker and easier for a Python interpreter to find specific elements, as compared to a list.

```
pet_dictionary = {"dogs": ["Yorkie", "Collie", "Bulldog"], "cats":
["Persian", "Scottish Fold", "Siberian"], "rabbits": ["Angora", "Holland
Lop", "Harlequin"]}


print(pet_dictionary.get("dogs", 0))
# Should print ['Yorkie', 'Collie', 'Bulldog']
```

Lists only:

- are ordered sets;
- access list elements by index positions;
- require that these indices be integers;
- use square brackets [ ];
- use commas to separate each list element.

```
pet_list  = ["Yorkie", "Collie", "Bulldog", "Persian", "Scottish Fold",
"Siberian", "Angora", "Holland Lop", "Harlequin"]


print(pet_list[0:3])
# Should print ['Yorkie', 'Collie', 'Bulldog']
```

Feature by Data Structure	Dictionary	Set	List	String	Tuple

Definition	Stores key:value pairs	An unordered collection of unique elements	A sequential, mutable collection of any data type	A sequential, immutable collection of textual data	A sequential, immutable collection of any data type
Representation	{ 'a':[42], 'b':[23,6,1] }	{'^2', 'mc', ' equal', 'E'}	['a','b', 3, 4]	“call me ishmael”	('commander','lambda')
How to create?	x = {}, x = dict()	x = set()	x = [], x = list()	x = (), x = str()	x = (‘a’,’b’,), x=tuple()
Is structure mutable and allow duplicate elements?	immutable keys but mutable and duplicate values	mutable but unique elements only	mutable and allows duplicate elements	immutable but allows duplicate elements	immutable but allows duplicate elements
Is the structure iterable?	no, it is unordered and random	no, it is unordered and unique	yes, and with numeric index assignment	yes, but with a sequence of textual data	yes, and with numeric index assignment

Code format examples

A recursive function will usually have this structure:

```
1  def recursive_function(parameters):
2      if base_case_condition(parameters):
3          return base_case_value
4      recursive_function(modified_parameters)
```

```
def recursive_function(parameters):
```

```
    if base_case_condition(parameters):
        return base_case_value
    recursive_function(modified_parameters)
```

General format

You can create a list from an iterable using a for loop, which is a useful way to iterate over an iterable:

```
1   new_list = []
2   for thing in list_of_things:
3       new_list.append(do_something(thing))
```

```
new_list = []
for thing in list_of_things:
    new_list.append(do_something(thing))
```

But do this instead

```
new_list = []
for thing in list_of_things:
    new_list.append(do_something(thing))
# Create a list of tuples where each tuple contains the numbers 1, 2, and
3.
numbers = [(1, 2, 3) for _ in range(5)]
```

Classes

```
class ClassName:
    """Documentation for the class."""
    def method_name(self, other_parameters):
        """Documentation for the method."""
        body_of_method

def function_name(parameters):
    """Documentation for the function."""
    Body_of_function
```