Contents

[Introduction To Python 4](#_Toc37507527)

[Introduction to Script 4](#_Toc37507528)

[Python Environment 4](#_Toc37507529)

[Getting Started 4](#_Toc37507530)

[Operators 5](#_Toc37507531)

[Flow Controlr 5](#_Toc37507532)

[String Handling 6](#_Toc37507533)

[Collections 6](#_Toc37507534)

[List 6](#_Toc37507535)

[Tuple 7](#_Toc37507536)

[Sets 7](#_Toc37507537)

[Dictionary 7](#_Toc37507538)

[Functions 8](#_Toc37507539)

[Lambda functions 8](#_Toc37507540)

[Modules 8](#_Toc37507541)

[OOPS concept 9](#_Toc37507542)

[Exception Handling 10](#_Toc37507543)

[File Handling 10](#_Toc37507544)

[Regular Expression 11](#_Toc37507545)

[Database access 11](#_Toc37507546)

[Advances Python 12](#_Toc37507547)

[Data Analytics 12](#_Toc37507548)

[Python Web Development 13](#_Toc37507549)

[Python Basics 15](#_Toc37507550)

[Rules 15](#_Toc37507551)

[Data Types 15](#_Toc37507552)

[How to declare variable 15](#_Toc37507553)

[Example Program 16](#_Toc37507554)

[How to take Input from end user 16](#_Toc37507555)

[Dynamic Typing 17](#_Toc37507556)

[Strong Typing 17](#_Toc37507557)

[PPS model in Python 17](#_Toc37507558)

[Operator 17](#_Toc37507559)

[Numeric Operator 17](#_Toc37507560)

[Comparison Operator 17](#_Toc37507561)

[Arithmetic Operator 17](#_Toc37507562)

[Logical operator 17](#_Toc37507563)

[Operations on Sequence 18](#_Toc37507564)

[Print format 18](#_Toc37507565)

[Conditional Loop 18](#_Toc37507566)

[If condition 18](#_Toc37507567)

[String Objects 20](#_Toc37507568)

[String operation method 20](#_Toc37507569)

[Tuple 20](#_Toc37507570)

[Sets 20](#_Toc37507571)

[Dictionary 20](#_Toc37507572)

[Map 20](#_Toc37507573)

[Reduce 20](#_Toc37507574)

[Filter 20](#_Toc37507575)

[OOPS concept 20](#_Toc37507576)

[Background 20](#_Toc37507577)

[**What Is Object-Oriented Programming (OOP)** 20](#_Toc37507578)

[**Why we need class** 21](#_Toc37507579)

[**OOPS Principle** 21](#_Toc37507580)

[**How to define a class** 22](#_Toc37507581)

[**Creating Objects** 22](#_Toc37507582)

[Class Variables 23](#_Toc37507583)

[**Instance Variables** 24](#_Toc37507584)

[Parameters 25](#_Toc37507585)

[Local Variables 25](#_Toc37507586)

[Defining methods 25](#_Toc37507587)

[Difference between function and methods 25](#_Toc37507588)

[ Python Objects (Instances) 25](#_Toc37507589)

[ How To Define a Class in Python 25](#_Toc37507590)

[Instance Attributes 25](#_Toc37507591)

[The \_\_init\_\_ method 25](#_Toc37507592)

[Class Attributes 26](#_Toc37507593)

[ Instantiating Objects 26](#_Toc37507594)

[What’s Going On? 26](#_Toc37507595)

[Review Exercises (#1) 26](#_Toc37507596)

[ Instance Methods 26](#_Toc37507597)

[Modifying Attributes 26](#_Toc37507598)

[ Python Object Inheritance 26](#_Toc37507599)

[Inheritance 26](#_Toc37507600)

[Encapsulation 28](#_Toc37507601)

[Polymorphism 30](#_Toc37507602)

[Review Exercise 31](#_Toc37507603)

[Bad Practices 31](#_Toc37507604)

[Working with files 31](#_Toc37507605)

[Conversion function 32](#_Toc37507606)

[Python Advances 32](#_Toc37507607)

[Modules 32](#_Toc37507608)

[Exception Handling 32](#_Toc37507609)

[Database programming 32](#_Toc37507610)

[Numpy 33](#_Toc37507611)

[Pandas 33](#_Toc37507612)

[Data Visualization(Matplotlib, seaborn, Plotly, Cuflinks) 34](#_Toc37507613)

[Projects 34](#_Toc37507614)

[Web Scrapping 34](#_Toc37507615)

[Image scrapping with deployment using Flask 34](#_Toc37507616)

[Machine Learning 34](#_Toc37507617)

[Linear regression 34](#_Toc37507618)

[Logistics regression 34](#_Toc37507619)

[Neutral networks 34](#_Toc37507620)

[CNN 34](#_Toc37507621)

[YOLO 34](#_Toc37507622)

[Hands on above topics 34](#_Toc37507623)

# Introduction To Python

What is Python

Python use cases

The Birth of Python

Python Timelines

Features of Python

Version of Python

Python distributors

# Introduction to Script

What is script

What is program

Types of script

Types of Programming languages

# Python Environment

Installation of Python on Windows

Settings path for Python

Python documentation

Getting help

Python command line shell, Editors and IDE

Basic Syntax

Running Python script on windows

Working with IDLE

# Getting Started

Keyword vs Data types

Static Data types version Dynamic data types

Fundamental Data types

Collection types

Number Systems

Mutable Objects versus Immutable Objects

Variables

Naming convetions

Print(), type() and id() functions

Input and raw\_input() functions

Type conversion function

# Operators

AIrthmetic operators

Relational operators

Logical operators

Short hand Assignement operators

Bitwise Operators

Membership operators

Identity Operators

Precedence of Operators

Evaluating expressions

# Flow Controlr

About Flow control

Elements of Flow control

Block/Cause

Conditinal Statements

Simple if

If else

If elif else

Looping Statements

While loop

While else

For loop

For else

Using range in for loop

Working with infinite loop

Breaks statement

Contitnue statement

Pass statement

# String Handling

What is String

Single quoted String literal

Triple quoted string literal

String indexing

String slicing

Working with String functions

Working with String methods

Reversing a String

String manipulation and concatenation

# Collections

## List

Lists are mutable

Getting to lists

List indices

Traversing a list

List operations

Lis slices

List methods

Map, filter and reduce

Deleting elements

List and strings

## Tuple

Advantage of tuple over list

Packing and unpacking

Comparing Tuples

Creating nested tuple

Using tuples as key in dictionaries

Deleting tuples

Slicing tuples

Tuple membership test

Buil-in function with tuple

Dotted charts

## Sets

How to create sets

Iteration over sets

Python set methods

Python set operations

Union of sets

Build in function with set

Python Frozenset

## Dictionary

How to create dictionary

Python hashing

Python dictionary methods

Copying dictionary

Updating dictionary

Delete key from the dictionary

Dictionary items() method

# Functions

Defining a function

Calling a function

Function parameters

Types of parameters

Default parameters

Non default parameters

Key word arguments

Non key word arguments

Arbitrary arguments

Return statement in functions

Handling return values

Global variable and local variable

Scope of global variable and local variable

Passing collection to a function

# Lambda functions

Lambda functions / anonymous functions

Filter()

Map()

Reduce()

# Modules

What is module

Types of module

The import statement

Module Aliases/renaming in module

From import

Reloading a module

Build in properties of a module

Dir() function

Creating user defined modules

Module search path

Command line arguments

Working with pre defined standard modules (Math, random, Datetime, Os, Sys, String ..)

Packages

Introduction to package

\_\_init\_\_.py file

Defining packages

Importing from packages

Defining sub packages

Importing from sub packages

# OOPS concept

Introudcion to OOPS programming

OOPS principle

Encapsulation

Defining classes

Creating objects

Class variables

Instance variables

Parameters

Local variable

Defining methods

Difference between functions and method

Instance method

Static method

Class method

Difference between static and class method

Constructors

Destructors

Inheritance

Types of inheritance

Polymorphism (overloading and overriding)

Super statement

Build in properties of a class

Inner class

# Exception Handling

Syntax error

Runtime Errors

What is Bug

What is exception

Need of exception handling

Predefined exceptions

Predefined exception hierarchy

Try, except, and finllay clauses

Named except block

Default except block

Handling multiple exceptions

Nested try, except and finally blocks

User defined exceptions

Raise, assert statements

# File Handling

What is a file

Opening a file

Reading data from a file

Writing data to a file

Closing a file

Working with methods of file object

Replacing the content of file

Working with directories

Handling IO exceptions

# Regular Expression

Introduction to regular expression

Simple character matches

Special characters

Character classes

Quantifiers

Forming regular expression

Matching at beginning or end

Greedy matched

Compiling regular expressions

Grouping

Match objects

Match(), search and sub() functions

Matching versus searching

Splitting a string

Replacing text

Flags

# Database access

Introduction to RDBMS

Cursor objects

Executing SQL queries

Executiong SQL queries with bind variables

Execution of PL/SQL procedure and functions

Installation of mysql database

Creating database in mysql

Creating users and assigning prvileges to the user in mysql

Installing of mysql python modules

Establishing connection with mysql

Clsoign mysql database connection

Execution of insert, update, delete and select queries

Handling db errors

# Advances Python

Python iterator

Python generator

Python closure

Python decorators

Python property

PIP

Installing of external modules using PIP

Working with CSV, XML and json files

Debugging using IDE

Test cases implementation

# Data Analytics

Introduction to big data

Pandas

Numpy Matplotlib

Hadoop word count example

Python Web Development

# Python Web Development

Introduction to Web application

Architecture of web application

Introduction to DJANGO framework

Introduction to FLASK framework

Introduction to web development

Introduction to Django

MVT pattern

Flow in Django project

Virtualenv

Installing Django

Creating Django project

What is view

Functions based views

Class based views

Http response

URL route configuration

CSRF

HTML and Django

About templates

Django template syntax

Static files

Loading templates

Context in Templates

Sending data

Forms Overview

Get and Post

The form class

Processing the form

Validation

Forms in templates

Database Setup

Creating models

Activation models

Activating modes

Admin site

One to many relation

Many to one relation

One to one relation

Pagination

Generic views

List views

Detail view

Create view

Update view

Delete view

Cookies in Django

Django session framework

Caching

File uploading and downloading

Deployment on Apache

Sending email

Other email functions

Security with user authentication

Automated testing

When to create tests

Using Django’s test framework

Using the test client

Checking code coverage

# Python Basics

## Rules

1. There are not semi colons
2. There are no curly braces for making start /end of a module/if/loop
3. We do not declare variables
4. For single line comment we use “#” symbol for a single line comment
5. Does not support multi line comment
6. A newline terminates each individual statement
7. You can also use semicolon to separate statement example principal =1000 ; rate = 0.05
8. printing to console/reading from keyboard the datatypes which is associated with the value is always a string

## Data Types

* number - int, float
* string - a collection of chars
* Boolean - true/false

### How to declare variable

*# No need to declare data types with variable*var = 32 *# treated as integer*print (var)  
print (type(var)) *# prints data type*dbl = 1.5 *# float variable*print (dbl)  
print (type(dbl)) *# prints data type*str = **"hello"**print (str) *# String data*print (type(str)) *# prints data type*isTrue = **False**print (isTrue) *# boolean data*print (type(isTrue)) *# prints data type*

print(id(a)) – id keyword prints memory address

## Example Program

### How to take Input from end user

var = input("Enter the number:")

print(type(var))

Q1. To calculate area of circle where radius is provided by the end user

- ask user to enter radius

- store it into var rad

- calculate area using the formula area = 3.141\*rad\*rad

- print a read

Q2. Write down a program which takes user input for principle, rate and time, based on user input calculate the simple interest the user has to pay

principal-int

rate-double

time-int

SI = (P\*R\*T)/100

### Dynamic Typing

### Strong Typing

### PPS model in Python

# Operator

## Numeric Operator

The following operations can be applied to all numeric types -

* x + y addition
* x – y subtraction
* x \* y multiplication
* x / y division
* x \*\* y power of x
* x % y Modulo (x mod y)
* --x unary minus
* +x unary plus

Build in functions on all numeric types

* Abs(x) – absolute value
* divmod(x ,y) - return (int ( x / y), x % y)
* pow (x , y [, modulo] – returns (x \*\* y) % modulo
* round (x, [n]) – returns to the nearest integer (floating point number only)

## Comparison Operator

* x < y – less than
* x > y – greater than
* x == y – equal to
* x != y – not equal to
* x >= y – greater than or equal to
* x <= y – less than or eaual to

## Arithmetic Operator

## Logical operator

X or y – Logical or - if x is false, return Y, otherwise return x

X and y – Logical and - if x is false, return x, otherwise return y

Not x - logical negation : if x is false, return 1, otherwise 0

## Operations on Sequence

The following operators can be applied to sequence types including strings, lists and tuples :

* s + r – concatenation
* s \* n – makes n copies of s where n is an integer
* s % d – String formatting (strings only)
* s[i] – indexing
* s [ I : j ] – slicing only
* x in s, x not in s – membership
* for x in s: - iteration
* len(s) – length
* min(s) – Minimum item
* max(s) – maximum item

## Print format

# Conditional Loop

## If condition

a=10  
b=20  
  
*####################################################  
#Simple If test  
####################################################***if** a<b:  
 z=b  
**else**: *# else is optional* z=a  
print (z)  
  
*####################################################  
#To create an empty clause, use the pass statement  
####################################################*a=11  
b=12  
t = 0  
*# Simple if test***if** a<b:  
 **pass** *#To create an empty clause, use the pass statement***else**:  
 t=a  
print (t)  
  
*####################################################  
# Form Boolean expression using or, and and not keywords  
####################################################*c = 5  
**if** b >= a **and** b <=c:  
 print (**"b is between a and c"**)  
**if not** (b < a **or** b > c):  
 print (**"b is still between a and c"**)  
  
*####################################################  
# Use elif statment to handle multiple tests  
####################################################*a = **"\*"  
if** a == **'+'**:  
 op = **"PLUS"  
elif** a == **'-'**:  
 op = **"MINUS"  
elif** a == **'\*'**:  
 op = **"MULTIPLY"  
else**:  
 **raise** RuntimeError(**"Unknow operator"**)  
print (op)

If condition with else

For loop

For I in s :

Statements

For loop with else

Break

Continue

While

Range function

## String Objects

### String operation method

List objects

# Tuple

# Sets

# Dictionary

# Map

# Reduce

# Filter

## OOPS concept

### Background

#### In this video, we will learn about the OOPs concept in Python.

Please be informed that OOP is a programming paradigm and not a Python concept. So the good news is that learning object-oriented programming fundamentals will be valuable to you even you are not working in Python.

### **What Is Object-Oriented Programming (OOP)**

Object-oriented programming is an approach for modeling real-world things as software entities like cars as well as relations between things like companies and employees, students and teachers, etc.

Every object has two things - attribute and behavior. When I say attribute, you can imagine it as data or properties of an object. Lets take an example of car object -

* Color: red, green, silver, brown
* Style: sport, standard , hybrid
* Make: Honda, BMW, Suzuki, TATA
* Doors
* wheels

When I say **object behavior**, this defines how an object will react" if asked to do something by another class or object or if its internal attribute changes. Behavior is the only way objects can do anything to themselves or have anything done to them. Example -

* Start the engine
* Stop the engine
* Speed up
* Change gear
* Apply brakes

In order to create object, you need to define class. A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made.

### **Why we need class**

Python has multiple primitive data structures , like numbers, strings, and lists which are designed to represent simple things like the cost of something, the name of a poem, and your favorite colors, respectively. These primitive data types does not support storing complex data.

### **OOPS Principle**

**Inheritance** - Inheritance facilitates reusability without modifying existing class. It is a mechanism in which one class acquires the property of another class. For example, a sports car class child inherits the attribute and behavior of vehicle class. With inheritance, we can reuse the attributes and behavior of the existing class.

**Encapsulation** - It describes the idea of bundling data and methods that work on that data within one unit. This concept is also often used to hide the internal representation, or state, of an object from the outside.

**Polymorphism** - A concept of using common operation in different ways for different data input. More specifically, it is the ability to redefine methods in child classes.

### **How to define a class**

*# Defining a class  
  
#You start with the class keyword to indicate that you are creating a class  
#then you add the name of the class (using CamelCase notation, starting with a capital letter.)***class** Car:  
 **pass** *# Used Python keyword pass here. This is very often used as a place holder where code will eventually go.  
 # It allows us to run this code without throwing an error.*

**Output:**

### **Creating Objects**

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

Class objects support two kinds of operations: attribute references and instantiation.

The example for object of parrot class can be:

obj = Test()

*#Create a class***class** Car:  
 color = **"red"  
  
 def** startTheEngine(self):  
 print(**"Engine started"**)  
  
*#Create a object*t1 = Car()  
  
*#Print attribute of a class*print(t1.color)  
  
*#call function*t1.startTheEngine()

### Class Variables

Class variables are variables whose value is assigned in class.

# Python program to show that the variables with a value

# assigned in class declaration, are class variables and

# variables inside methods and constructors are instance

# variables.

# Class for Computer Science Student

class CSStudent:

    # Class Variable

    stream = 'cse'

    # The init method or constructor

    def \_\_init\_\_(self, roll):

        # Instance Variable

        self.roll = roll

# Objects of CSStudent class

a = CSStudent(101)

b = CSStudent(102)

print(a.stream)  # prints "cse"

print(b.stream)  # prints "cse"

print(a.roll)    # prints 101

# Class variables can be accessed using class

# name also

print(CSStudent.stream) # prints "cse"

### **Instance Variables**

In Python, instance variables are variables whose value is assigned inside a constructor or method with self.

We can define instance variables inside normal methods also.

# Python program to show that we can create

# instance variables inside methods

# Class for Computer Science Student

class CSStudent:

# Class Variable

stream = 'cse'

# The init method or constructor

def \_\_init\_\_(self, roll):

# Instance Variable

self.roll = roll

# Adds an instance variable

def setAddress(self, address):

self.address = address

# Retrieves instance variable

def getAddress(self):

return self.address

# Driver Code

a = CSStudent(101)

a.setAddress("Noida, UP")

print(a.getAddress())

### Parameters

### Local Variables

### Defining methods

### Difference between function and methods

###  Python Objects (Instances)

###  How To Define a Class in Python

### Instance Attributes

The \_\_init\_\_ method  
The \_\_init\_\_ method is similar to constructors in and Java. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

# A Sample class with init method

class Person:

# init method or constructor

def \_\_init\_\_(self, name):

self.name = name

# Sample Method

def say\_hi(self):

print('Hello, my name is', self.name)

p = Person('Shwetanshu')

p.say\_hi()

Output:

Hello, my name is Shwetanshu

Here, we define the \_\_init\_\_ method as taking a parameter name (along with the usual self). .

### Class Attributes

###  Instantiating Objects

### What’s Going On?

### Review Exercises (#1)

###  Instance Methods

### Modifying Attributes

###  Python Object Inheritance

## Inheritance

Inheritance is a way of creating new class for using details of existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

# parent class

class Bird:

def \_\_init\_\_(self):

print("Bird is ready")

def whoisThis(self):

print("Bird")

def swim(self):

print("Swim faster")

# child class

class Penguin(Bird):

def \_\_init\_\_(self):

# call super() function

super().\_\_init\_\_()

print("Penguin is ready")

def whoisThis(self):

print("Penguin")

def run(self):

print("Run faster")

peggy = Penguin()

peggy.whoisThis()

peggy.swim()

peggy.run()

Output

In the above program, we created two classes i.e. Bird (parent class) and Penguin (child class). The child class inherits the functions of parent class. We can see this from swim() method. Again, the child class modified the behavior of parent class. We can see this from whoisThis() method. Furthermore, we extend the functions of parent class, by creating a new run() method.

Additionally, we use super() function before \_\_init\_\_() method. This is because we want to pull the content of \_\_init\_\_() method from the parent class into the child class.

## Encapsulation

Using OOP in Python, we can restrict access to methods and variables. This prevent data from direct modification which is called encapsulation. In Python, we denote private attribute using underscore as prefix i.e single “ \_ “ or double “ \_\_“.

class Computer:

def \_\_init\_\_(self):

self.\_\_maxprice = 900

def sell(self):

print("Selling Price: {}".format(self.\_\_maxprice))

def setMaxPrice(self, price):

self.\_\_maxprice = price

c = Computer()

c.sell()

# change the price

c.\_\_maxprice = 1000

c.sell()

# using setter function

c.setMaxPrice(1000)

c.sell()

In Python, we use double underscore (Or \_\_) before the attributes name and those attributes will not be directly visible outside.

class MyClass:

      # Hidden member of MyClass

    \_\_hiddenVariable = 0

    # A member method that changes

    # \_\_hiddenVariable

    def add(self, increment):

        self.\_\_hiddenVariable += increment

        print (self.\_\_hiddenVariable)

# Driver code

myObject = MyClass()

myObject.add(2)

myObject.add(5)

# This line causes error

print (myObject.\_\_hiddenVariable)

## Polymorphism

Polymorphism is an ability (in OOP) to use common interface for multiple form (data types). Suppose, we need to color a shape, there are multiple shape option (rectangle, square, circle). However we could use same method to color any shape. This concept is called Polymorphism.

class Parrot:

def fly(self):

print("Parrot can fly")

def swim(self):

print("Parrot can't swim")

class Penguin:

def fly(self):

print("Penguin can't fly")

def swim(self):

print("Penguin can swim")

# common interface

def flying\_test(bird):

bird.fly()

#instantiate objects

blu = Parrot()

peggy = Penguin()

# passing the object

flying\_test(blu)

flying\_test(peggy)

### Review Exercise

## Bad Practices

Protect the abstraction

Def \_\_init\_\_(self, \*args) – takes variable number of arguemtns

Def \_\_init\_\_(self, \*\*args) – takes variable number of dictionary

Super() keyword to initialize parent class

### Working with files

1. Open the file

F = open(“file path”, mode=’’r’ encoding=’cp1242’)

Windows support Cp1242, Linux system – UTF\*

1. Read the file

f.read()

1. Close the file

f.close()

try:

F = open(“file path”, mode=’’r’ encoding=’cp1242’)

#perform read operation on file

Finally:

f.close()

**you can use loop file**

with open(“Read.txt”,’w’ encoding ‘uft-8) **as** f:

f.write(“1”)

f.write(“2”)

f.write(“3”)

Note – as means alias name

For line in f:

Print(line)

F.seek() method

How to do read and write function on a file?

# Conversion function

this method convert one data type to another

* int() - convert string to int
* float() - convert string to float
* str() - converts number/bool to string

Example :

var = int(input("enter the number : ")) - first input will be read from the keyboard and then converted to integer value and assigned to the var

# Python Advances

## Modules

## Exception Handling

## Database programming

By default sqllite database is there

1. first create database

import sqlite3 as s

#connecting with database

Db = s.connect(“my\_db.db”);

#drop table

Db.execute(“drop table if exists grades1”)

#create table as per requirement

Db.execute(“create table grades1(id int,

Results = db.execute(“select \* from grades1 order by id”)

Print(Type(results))

For row in results:

Print (row)

Print(“-“ \* 60)

# Numpy

# Pandas

Built on top of numPy

What is Panda

**Series** – read one column like list

**DataFrame** – read row and column

Operation supported by DataFrame

Read data from multiple sources like CSV, excel, html or database

Type method

Import pandas as pd

Path =”c”\a.csv”

Draft1.read\_csv(path)

Draft1.read\_csv(path, header=None)

Draft.head(4)

Read\_table

How to read specific column from CSV file

Read\_html

# Data Visualization(Matplotlib, seaborn, Plotly, Cuflinks)

# Projects

## Web Scrapping

## Image scrapping with deployment using Flask

# Machine Learning

## Linear regression

## Logistics regression

## Neutral networks

## CNN

## YOLO

# Hands on above topics

PASS - The **pass keyword** as name suggests, does nothing. It is used as a dummy place holder whenever a syntactical requirement of a certain programming element is to be fulfilled without assigning any operation. In other words, the **pass** statement is simply ignored by the **Python** interpreter and can be seen as a null statement

Self keyword – it refers to object itself. It should be first parameter in method.

Jitsi.org

25 code snippet -

<https://medium.com/better-programming/25-useful-python-snippets-to-help-in-your-day-to-day-work-d59c636ec1b>

tic tac toc game -

<https://medium.com/better-programming/tic-tac-toe-with-python-8fb6d666b13f>

Build email assistant

<https://medium.com/swlh/i-used-python-to-make-a-personal-assistant-and-it-drafts-emails-for-me-434ac3ab6d38>