

INTRODUCTION TO PYTHON

DAY 2 OF 20



PREPARED BY DSEA

CONTENT

- What is python
- Python Quickstart
- Python Comments
- Intro to Python Variables

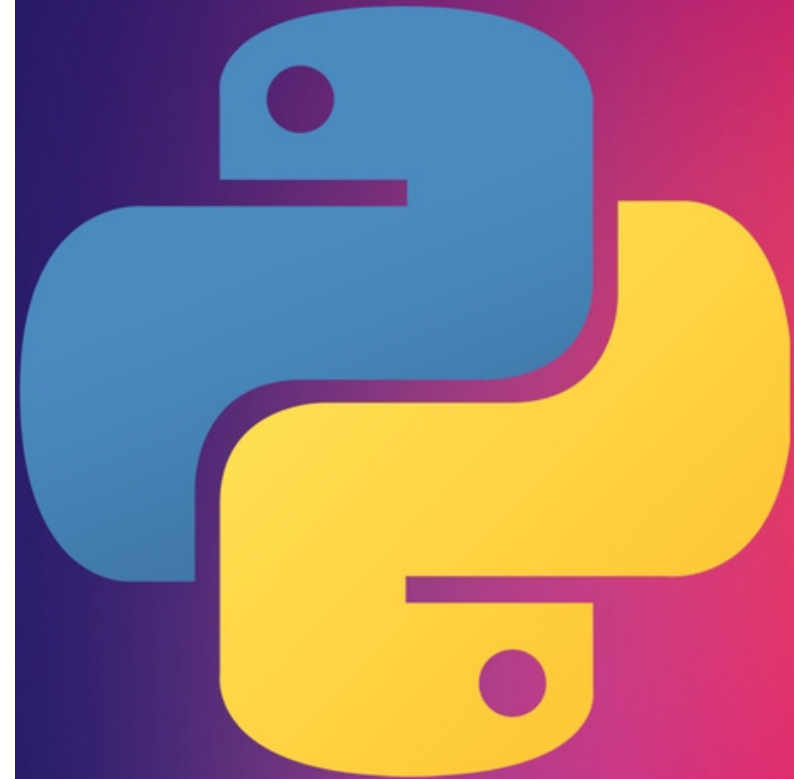
WHAT IS PYTHON

```
31 def __init__(self, path):
32     self.file = None
33     self.fingerprints = set()
34     self.logdupes = True
35     self.debug = debug
36     self.logger = logging.getLogger(__name__)
37     if path:
38         self.file = open(os.path.join(path, 'requests.log'), 'a')
39         self.file.seek(0)
40         self.fingerprints.update(self.read())
41
42     @staticmethod
43     def from_settings(settings):
44         debug = settings.getboolean('debug')
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```



WHAT IS PYTHON

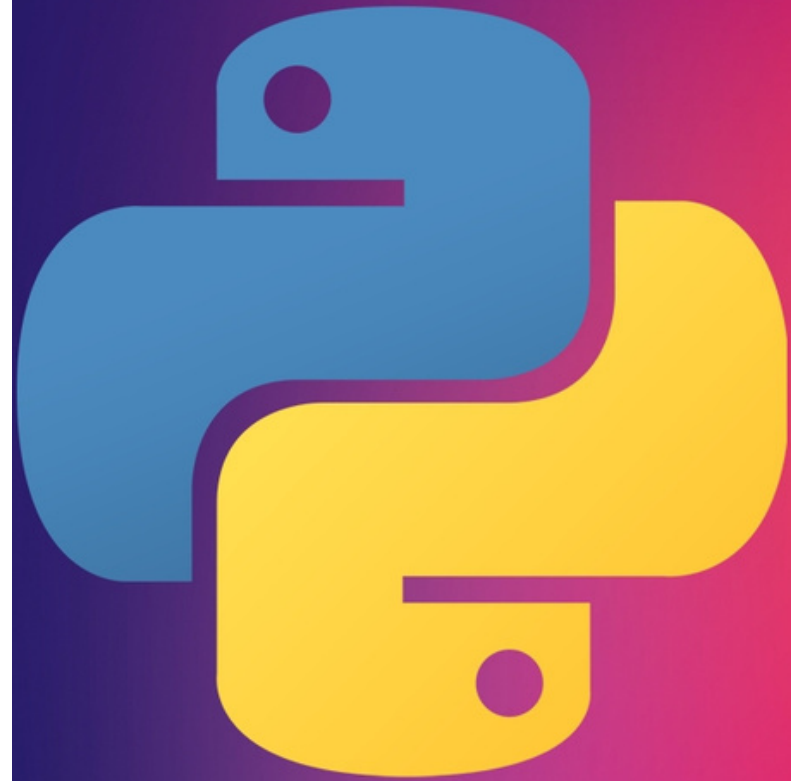
- Python is an interpreted, high-level, general-purpose programming language.
- Created by Guido van Rossum and first released in 1991,
- Python's design philosophy emphasizes **code readability** with its notable use of significant whitespace.
- Its language constructs and **object-oriented approach** aim to help programmers write clear, logical code for small and large-scale projects.





WHAT IS PYTHON

- Python is dynamically typed and garbage-collected.
- It supports multiple programming paradigms, including procedural, object oriented, and functional programming.
- Python is often described as a **"batteries included"** language due to its comprehensive standard library.
- Python is a **popular programming language**. It was created by Guido van Rossum, and released in 1991.





USES OF PYTHON

- Web development (server-side),
- Software Development,
- Mathematics,
- System Scripting.

```
self.file = None
self.fingerprints = set()
self.logdups = True
self.debug = debug
self.logger = logging.getLogger(__name__)
if path:
    self.file = open(os.path.join(path, 'fingerprint.log'), 'a')
    self.file.seek(0)
    self.fingerprints.update(set(self._get_fingerprints(path)))

@classmethod
def from_settings(cls, settings):
    debug = settings.getboolean('DEBUG')
    return cls(job_dir(settings.get('JOB_DIR', '/tmp/')))

def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
        return True
    self.fingerprints.add(fp)
    if self.file:
        self.file.write(fp + '\n')

def request_fingerprint(self, request):
    return request_fingerprint(request)
```



WHAT CAN PYTHON DO?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and
- modify files.
- Python can be used to handle big data and perform complex
- mathematics.
- Python can be used for rapid prototyping, or for productionready software development.

```
self.file = None
self.fingerprints = set()
self.logdups = True
self.debug = debug
self.logger = logging.getLogger(__name__)
if path:
    self.file = open(os.path.join(path, 'fingerprint.log'), 'a')
    self.file.seek(0)
    self.fingerprints.update(fingerprint)

@classmethod
def from_settings(cls, settings):
    debug = settings.getboolean('DEBUG')
    return cls(job_dir(settings.get('JOB_DIR', '/tmp/')))

def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
        return True
    self.fingerprints.add(fp)
    if self.file:
        self.file.write(fp + '\n')

def request_fingerprint(self, request):
    return request_fingerprint(request)
```




WHY PYTHON?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has **syntax** that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an **interpreter** system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick

```
self.file = None
self.fingerprints = set()
self.logdups = True
self.debug = debug
self.logger = logging.getLogger(__name__)
if path:
    self.file = open(os.path.join(path, 'fingerprint.log'), 'a')
    self.file.seek(0)
    self.fingerprints.update(fingerprint)

@classmethod
def from_settings(cls, settings):
    debug = settings.getboolean('DEBUG')
    return cls(job_dir=settings.get('JOB_DIR'))

def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
        return True
    self.fingerprints.add(fp)
    if self.file:
        self.file.write(fp + '\n')

def request_fingerprint(self, request):
    return request_fingerprint(request)
```




WHY PYTHON?

- Python can be treated in a procedural way, an object orientated way or a functional way

```
self.file = None
self.fingerprints = set()
self.logdups = True
self.debug = debug
self.logger = logging.getLogger(__name__)
if path:
    self.file = open(os.path.join(path, 'fingerprint.log'), 'a')
    self.file.seek(0)
    self.fingerprints.update(request_fingerprint(request))

@classmethod
def from_settings(cls, settings):
    debug = settings.getboolean('DEBUG')
    return cls(job_dir=settings.get('JOB_DIR', '.'))

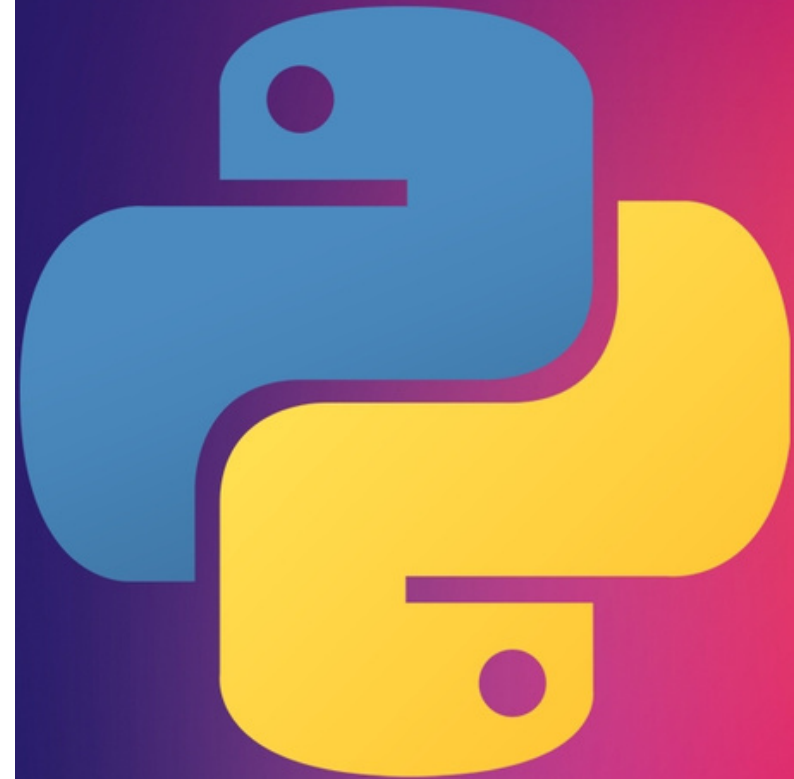
def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
        return True
    self.fingerprints.add(fp)
    if self.file:
        self.file.write(fp + '\n')


def request_fingerprint(self, request):
    return request_fingerprint(request)
```



SYNTAX COMPARED TO OTHER PROGRAMMING LANGUAGES

- Python was designed for **readability**, and has some similarities to the English language with influence from mathematics.
- Python uses **new lines** to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on **indentation**, using whitespace, to define scope; such as the scope of loops, functions and classes.
- Other programming languages often use curly-brackets for this purpose

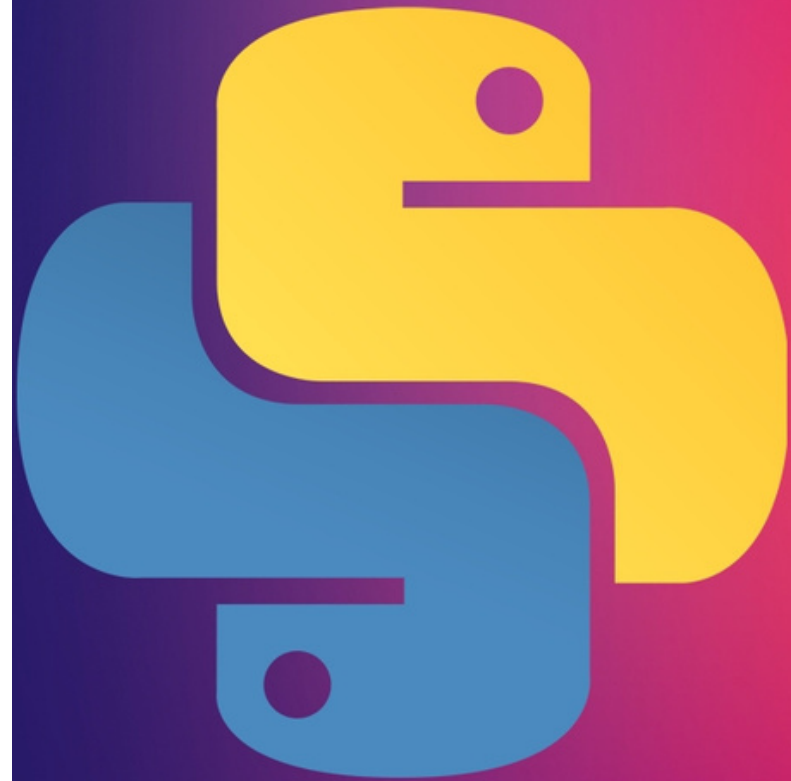


The background is a dark blue-to-purple gradient. In the center is a large, stylized Python logo with one snake in dark blue and the other in olive green. The text "PYTHON QUICKSTART" is written in a bold, cyan, sans-serif font across the middle of the logo. Various abstract geometric shapes are scattered around: a large red circle at the top center, a yellow 'X' on the left, a cyan 'X' at the top right, a yellow triangle at the top right, a red wavy line at the top right, a yellow wavy line at the bottom right, a red circle at the bottom left, and a cyan wavy line on the left.

PYTHON QUICKSTART

PYTHON IS AN INTERPRETED PROGRAMMING LANGUAGE

- This means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed
- Let's write our first Python file called **helloworld.py**, which can be done in any text editor.




```
def __init__(self):
    self.file = None
    self.fingerprints = set()
    self.logdupes = True
    self.debug = debug
    self.logger = logging.getLogger(__name__)
    if path:
        self.file = open(os.path.join(path, 'fingerprints.log'), 'a')
        self.file.seek(0)
        self.fingerprints.update(fingerprints)

    @classmethod
    def from_settings(cls, settings):
        debug = settings.getbool('debug')
        return cls(job_dir(settings), debug)

    def request_seen(self, request):
        fp = self.request_fingerprint(request)
        if fp in self.fingerprints:
            return True
        self.fingerprints.add(fp)
        if self.file:
            self.file.write(fp + os.linesep)

    def request_fingerprint(self, request):
        return request_fingerprint(request)
```



print("Hello,World!")

Congratulations,
you have written
your first Python program



PYTHON COMMENTS

```
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

def __init__(self, path):
    self.file = None
    self.fingerprints = set()
    self.logdupes = True
    self.debug = debug
    self.logger = logging.getLogger(__name__)
    if path:
        self.file = open(os.path.join(path, 'requests.log'),
                        'a')
        self.file.seek(0)
        self.fingerprints.update(e.request for e in self.log)

    @classmethod
    def from_settings(cls, settings):
        debug = settings.DEBUG
        return cls(settings.LOGGING_PATH, debug)

    def request_seen(self, request):
        fp = self.request_fingerprint(request)
        if fp in self.fingerprints:
            return True
        self.fingerprints.add(fp)
        if self.file:
            self.file.write(fp + os.linesep)

    def request_fingerprint(self, request):
        return request_fingerprint(request)
```



```
def __init__(self):
    self.file = None
    self.fingerprints = set()
    self.logdupes = True
    self.debug = debug
    self.logger = logging.getLogger(__name__)
    if path:
        self.file = open(os.path.join(path, 'fingerprint.log'), 'a')
        self.file.seek(0)
        self.fingerprints.update(self._load_fingerprints())

    @classmethod
    def from_settings(cls, settings):
        debug = settings.getbool('debug')
        return cls(job_dir(settings), debug)

    def request_seen(self, request):
        fp = self.request_fingerprint(request)
        if fp in self.fingerprints:
            return True
        self.fingerprints.add(fp)
        if self.file:
            self.file.write(fp + os.linesep)

    def request_fingerprint(self, request):
        return request_fingerprint(request)
```



COMMENTS

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.
- Creating a Comment
- Comments **starts with a #**, and Python will ignore them:



COMMENT

```
#this is a comment  
print("hello world")
```

Output will be hello world.

Comments can also be placed at the end of a line, and Python will ignore the rest of the line:

```
print("hello python") #this outputs hello python
```



COMMENT

Comments does not have to be text to explain the code, it can also be used to prevent Python from executing code.

```
#adding two numbers  
#print (sum of a and b)  
a=20  
b=30  
print(a+b)  
#output 50
```



MULTIPLE COMMENTS

Python does not really have a syntax for multi line comments.

To add a multi-line comment you could insert a `#` for each line, like the above example.

Also, since Python will ignore string literals that are not assigned to a variable, you can add a multi-line string (triple quotes) in your code, and place you comment inside it:

```
"""
```

```
    THIS IS FIRST LINE COMMENT
```

```
    THE SECOND LINE COMMENT
```

```
    THIS IS A NOTHER COMMENT
```

```
"""
```

```
print("hello world")
```



MULTIPLE COMMENTS

As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multi-line comment.



PYTHON VARIABLES

```
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
        return True
    self.fingerprints.add(fp)
    if self.file:
        self.file.write(fp + os.linesep)

def request_fingerprint(self, request):
    return request_fingerprint(request)
```




PYTHON VARIABLES

- Variables are containers for storing data values.
- Unlike other programming languages, Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it

```
x=20  
y=40  
print(x)  
print(y)
```

```
self.file = None  
self.fingerprints = set()  
self.logdups = True  
self.debug = debug  
self.logger = logging.getLogger(__name__)  
if path:  
    self.file = open(os.path.join(path, 'fingerprint.log'), 'a')  
    self.file.seek(0)  
    self.fingerprints.update(  
        [fingerprint for fingerprint in request_fingerprint(request)]  
    )  
  
@classmethod  
def from_settings(cls, settings):  
    debug = settings.getboolean('DEBUG', False)  
    return cls(job_dir(settings.get('JOB_DIR', None)),  
               log_dir(settings.get('LOG_DIR', None)),  
               debug=debug)  
  
def request_seen(self, request):  
    fp = self.request_fingerprint(request)  
    if fp in self.fingerprints:  
        return True  
    self.fingerprints.add(fp)  
    if self.file:  
        self.file.write(fp + '\n')  
  
def request_fingerprint(self, request):  
    return request_fingerprint(request)
```



PYTHON VARIABLES

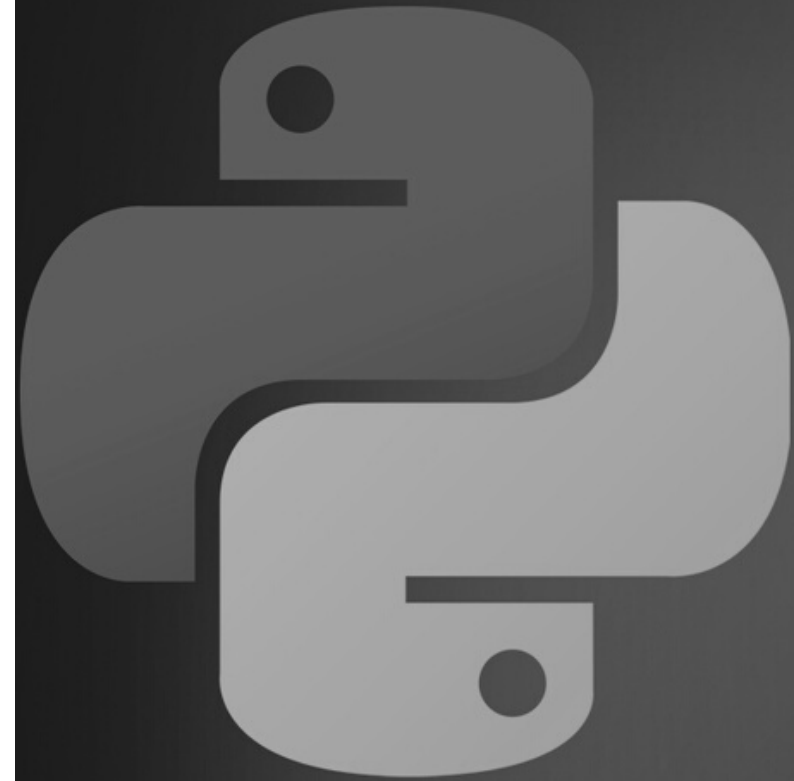
Variables do not need to be declared with any particular type and can even change type after they have been set.

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume)

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and





PYTHON VARIABLES

- Variable names are case-sensitive (age, Age and AGE are three different variables)
- **Remember that variable names are case-sensitive**

```
self.file = None
self.fingerprints = set()
self.logdups = True
self.debug = debug
self.logger = logging.getLogger(__name__)
if path:
    self.file = open(os.path.join(job_dir, path), 'a')
    self.file.seek(0)
    self.fingerprints.update(set(request_fingerprint(request).split(',')))

@classmethod
def from_settings(cls, settings):
    debug = settings.getboolean('DEBUG')
    return cls(job_dir=settings.get('JOB_DIR', '.'), log_level=settings.get('LOG_LEVEL', logging.INFO), debug=debug)

def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
        return True
    self.fingerprints.add(fp)
    if self.file:
        self.file.write(fp + '\n')

def request_fingerprint(self, request):
    return request_fingerprint(request)
```


Summary

NEXT TOPIC

PYTHON VARIABLES
PYTHON OPERATORS

-
-