Why should I learn Python?

Python is easy to read, easy to learn.

It's versatilie and simple.

You can create complex applications with a small amount of code

Note:

- It's incredibly powerful and versatile
- It's easy to read and understand
- There's not too many layers to it
  - You don't have to manually manage memeory
- It's commonly used
  - Currently the most popular programming language
  - It's used across the board

By the end of this session, you won't know everything

However, you'll have a decent head start at learning to code

Note:

- We're going to talk for the next 30-50 minutes
- then we're going to set you off onto a project in order to learn python

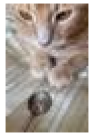
---

But what can you do in python

Image Manipulation

```
from PIL import Image

img = Image.open("input.png")
MAX_SIZE = (64, 64)
img.thumbnail(MAX_SIZE) # Scale down to 256x256
img.save("./output.jpeg") # Save as jpeg
```

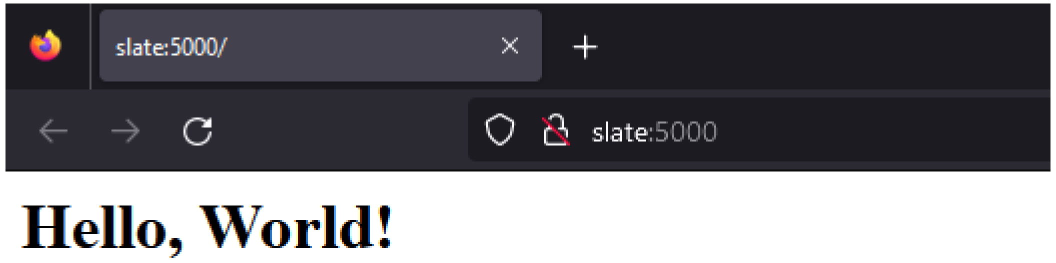


Websites

```
from flask import Flask

app = Flask(__name__)
```

```
@app.get("/")
def hello_world():
    return "<h1>Hello, World!</h1>"
```



Hello, World!

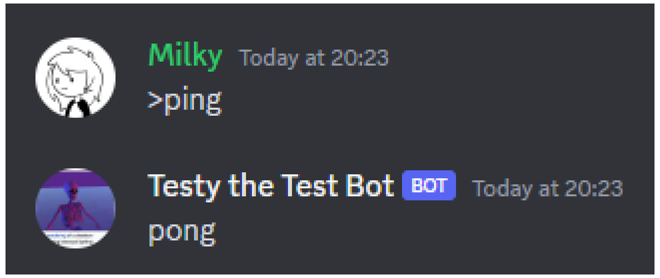
Discord Bot

```
import discord
from discord.ext import commands

intents = discord.Intents.default()
bot = commands.Bot(command_prefix='>', intents=intents)

@bot.command()
async def ping(ctx):
    await ctx.send('pong')

bot.run('SECRET TOKEN')
```



Milky Today at 20:23
>ping

Testy the Test Bot BOT Today at 20:23
pong

Interact with Web APIs

```python
import requests

r = requests.get("https://api.ipify.org?format=json")
data = r.json()
ip = data["ip"]
print("Your IP is", ip)
```

```
>>> Your IP is 41.163.35.16
```

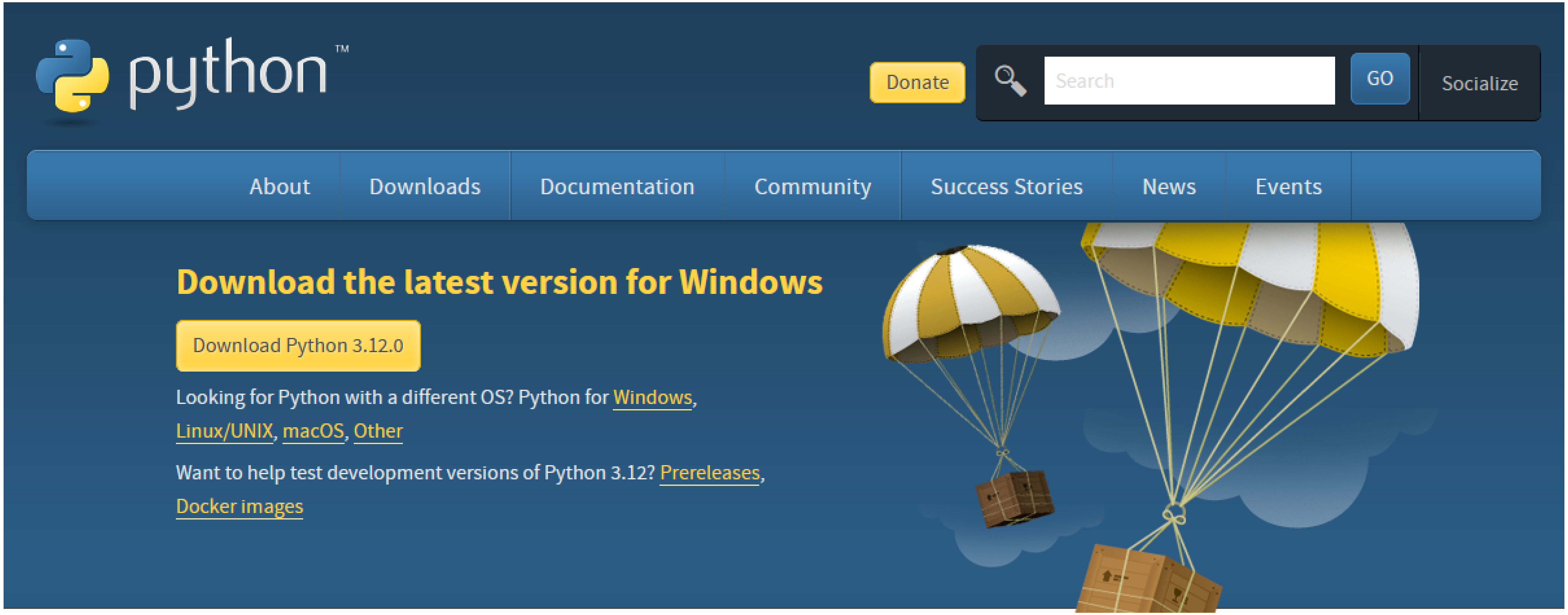
Setting Up

Note:

- For all those in the lab, this most likely won't apply to you
- However, for those watching at home, here's how to get set up

Install Python

https://www.python.org/downloads/



Download the latest version for Windows

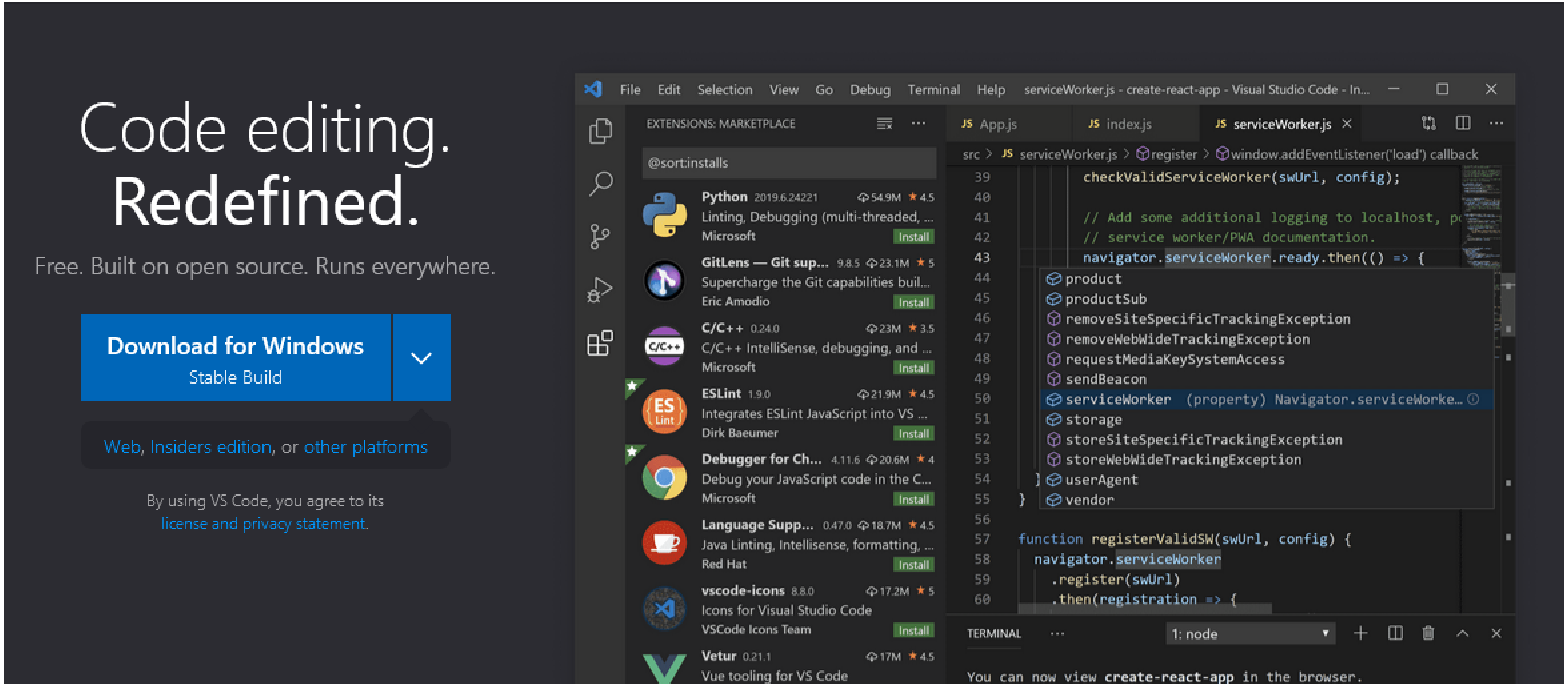
Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python 3.12? Prereleases, Docker images

---

Install VSCode

https://code.visualstudio.com/



Code editing. Redefined.

Free. Built on open source. Runs everywhere.

Download for Windows
Stable Build

Web, Insiders edition, or other platforms

By using VS Code, you agree to its license and privacy statement.

File
Edit
Selection
View
Go
Debug
Terminal
Help
serviceWorker.js - create-react-app - Visual Studio Code - In...
EXTENSIONS: MARKETPLACE
@sort:installs
App.js
index.js
serviceWorker.js
src > serviceWorker.js > register > window.addEventListener('load') callback

Python 2019.6.24221 54.9M ★4.5
Linting, Debugging (multi-threaded, ...
Microsoft Install

GitLens — Git sup... 9.8.5 23.1M ★5
Supercharge the Git capabilities buil...
Eric Amodio Install

C/C++ 0.24.0 23M ★3.5
C/C++ IntelliSense, debugging, and ...
Microsoft Install

ESLint 1.9.0 21.9M ★4.5
Integrates ESLint JavaScript into VS ...
Dirk Baeumer Install

Debugger for Ch... 4.11.6 20.6M ★4
Debug your JavaScript code in the C...
Microsoft Install

Language Supp... 0.47.0 18.7M ★4.5
Java Linting, Intellisense, formatting, ...
Red Hat Install

vscode-icons 8.8.0 17.2M ★5
Icons for Visual Studio Code
VSCode Icons Team Install

Vetur 0.21.1 17M ★4.5
Vue tooling for VS Code

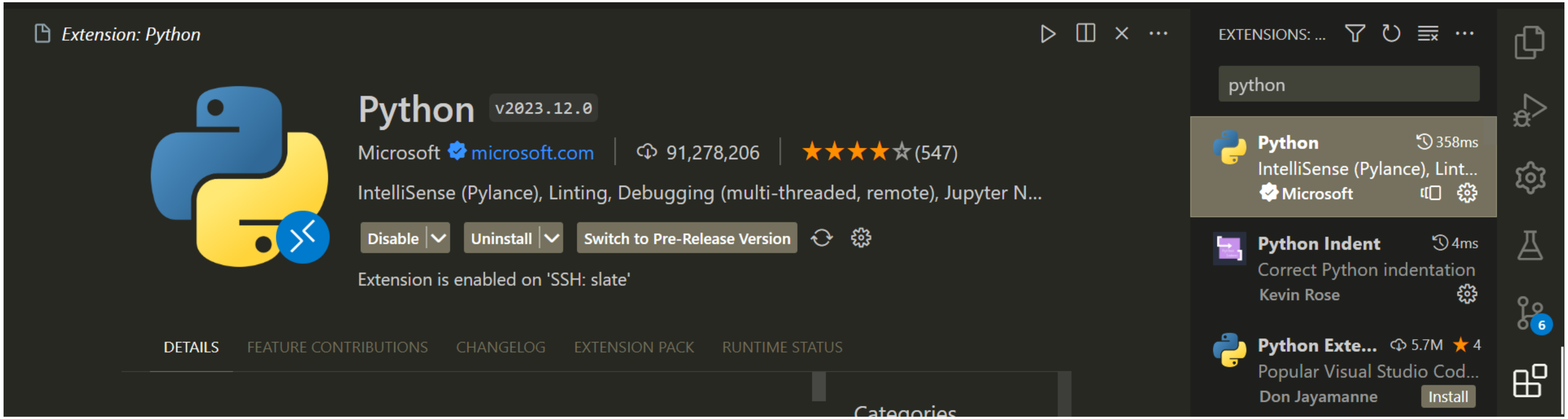
```
      checkValidServiceWorker(swUrl, config);

      // Add some additional logging to localhost, pc
      // service worker/PWA documentation.
      navigator.serviceWorker.ready.then(() => {
        product
        productSub
        removeSiteSpecificTrackingException
        removeWebWideTrackingException
        requestMediaKeySystemAccess
        sendBeacon
        serviceWorker  (property) Navigator.serviceWorke...
        storage
        storeSiteSpecificTrackingException
        storeWebWideTrackingException
    }   userAgent
  }     vendor

function registerValidSW(swUrl, config) {
  navigator.serviceWorker
    .register(swUrl)
    .then(registration => {
```

```
You can now view create-react-app in the browser.
```

Install Python Extensions



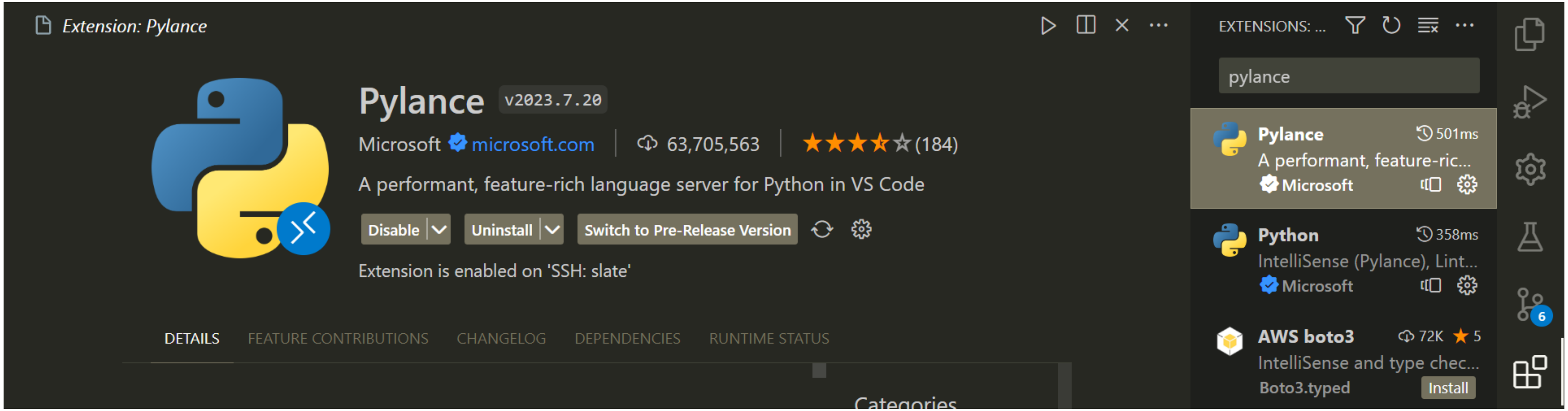
Extension: Python

Python v2023.12.0

Microsoft microsoft.com | 91,278,206 | ★★★★☆ (547)

IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), Jupyter N...

Extension is enabled on 'SSH: slate'



Extension: Pylance

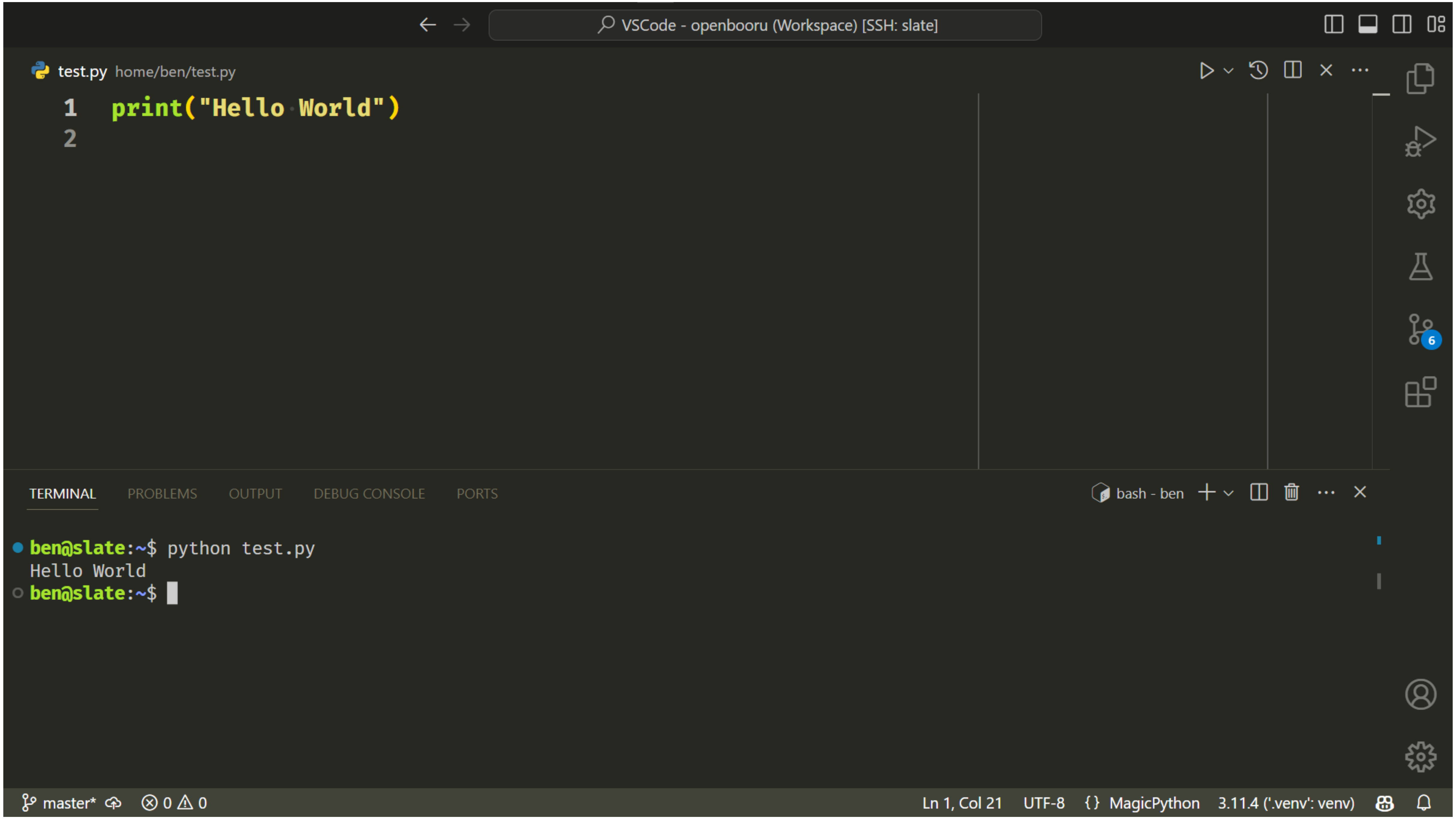
Pylance v2023.7.20

Microsoft microsoft.com | 63,705,563 | ★★★★☆ (184)

A performant, feature-rich language server for Python in VS Code

Extension is enabled on 'SSH: slate'

Hello World



```python
print("Hello World")
```

```
ben@slate:~$ python test.py
Hello World
ben@slate:~$ 
```

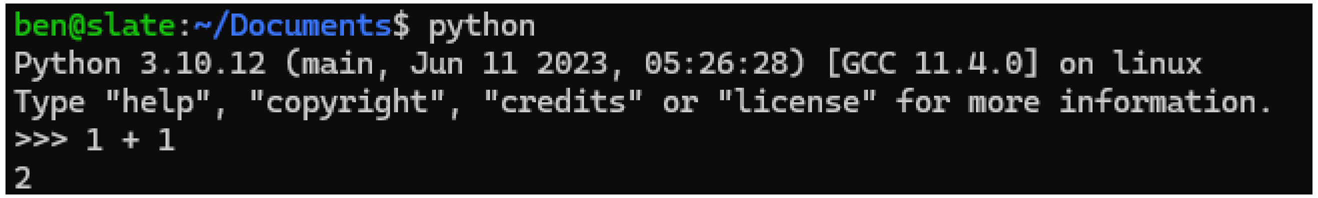
```python
print("Hello World")
```

Note:

- You can run a file by pressing the play button in the top left

---

REPL



```
ben@slate:~/Documents$ python
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 + 1
2
```

Note:

- Alternatively, if you'd want you can use the REPL
- This allows you to run python as you type it out

Fundamentals

- Variables
- Data Types
- Operations
- If Statements
- Loops
- Functions

Note:
Unspoken Rules

- Program flow goes from top to bottom
- Spacing is fine

Variables

```
pi = 3.14
print(pi)
> 3.14
```

Note:

- Variables are the bread and butter of programming
- One of the key fundamentals
- Variables act as a substitution

Variables work like a lookup table

```
one = 1
print(one)
> 1
```

Name	Value
one	1

They can also be reassigned

```
number = 1
print(number)
> 1
number = 2
print(number)
> 2
```

```
is_logged_in = False

if is_logged_in:
    message = "Welcome"
else:
        message = "Your not logged in"
```

Note:

- On it's own, this isn't very useful
- However, combined with ability to change code paths later makes it more important

---

Variable names are made up of letters, numbers, and underscores

All variables have to start with a letter or underscore

```
point = "Correct"
point_1 = "Yes"
_point = "Valid"
1_point = "Invalid"
# SyntaxError
+_symbol = "+"
# SyntaxError
```

Data Types

Every value has a type like `str`, `float`, `int`

It describes what type of data the value is

Name	Type	Description
Boolean	`bool`	True or False
Integer	`int`	Whole Number
Floating Point	`float`	Decimal Number
String	`str`	Text
List	`list`	List of values
Dictionary	`dict`	Table of keys and values
None	`None`	An empty value

Literals

```
boolean = True
number = 1
decimal = 1.5
text = ""
numbers_list = [1, 2, 3, 4, 5]
table = {
        1: "one",
        2: "two",
}
```

Note:

- This is an example of how to create them as what are called literals
- Literals means they're placed inside the code directly
- Rather than being created whilst the program is running

Boolean

```
is_water_wet = False
is_gravity_real = True
```

Note:

- Simpleist to understand
- Yes or No
- Talk about the capital letters

Int

```
answer_to_everything = 42
speed_of_light = 299_792_458
negative_one = -1
```

Note:

- Can also hold

Float

```
pi = 3.14
one = float(1)
```

Keep in mind, floats are inprecise. Don't use direct comparsion

```
> 0.2 + 0.2 + 0.2
0.6000000000000001
```

String

```
username = "ben"
dr_seuss = "Do you like green eggs and ham?"
```

Note:

- You can create a string by wrapping a piece of text in quotes
- Normal strings cannot go over multiple lines

You can make multi-line strings by using `"""` instead

```
PARAGRAPH = """
This is a big long paragraph,
that goes over multiple lines
:)
"""
```

Note:

- If you need to create a big string, you can use a multi line string

---

f-strings

Putting f in front of a string allows you to place variables inside

```
username = "Ben Brady"
print(f"Hello {username}")
> Hello Ben Brady
```

List

A list is known a collection type, it hold other values

```
prime_numbers = [2, 3, 5, 7, 11, 13]
random_junk = [1, "a", 1.0, []]
matrix = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9],
]
```

Note:

- It's the first mutable type
    - This means you can update it without reassigning it

List values can be accessed by indexing them

Watch out, lists indexes start at zero

```
prime_numbers = [2, 3, 5, 7, 11, 13]
third_prime = prime_numbers[2]
# 3rd prime number, not second
```

Your also allowed to update items

```
numbers = [1, 2, 3]
numbers[1] = 3
print(numbers)
> [1, 3, 2]
```

However, you can't add new values this way

```
numbers = [1, 2, 3]
numbers[3] = 3
# IndexError: list assignment index out of range
```

Instead, you have to use append

```
fruits = []
fruits.append("apple")
fruits.append("pear")
print(fruits)
> ["apple", "pear"]

fruits.append("strawberry")
print(fruits)
> ["apple", "pear", "strawberry"]
```

Note:

- This is the first method you'll see
- Most types have methods
  - They're unique per datatype
- They're the same as a function, accept they

Dictionary

Store a variable to table

```
table = {}
table["key"] = "value"
print(table)
> {'key': 'value'}
```

Note:

- A dict
- A key can be

A key has to be hashable / unchangeable

```
table = {}
table[1] = "one"
table[{}] = "dict"
# TypeError: unhashable type: 'dict'
```

Note:

- If you try to set a key a value

If you need to use multiple values, you can use a tuple

A tuple is like a list, but you can't change it

```
table[(0, 1)] = {}
```

None

```
value = None
```

Conversions

Some datatypes allows to convert to them

```
str(1)
> "1"
int("1")
> 1
float("3.14")
> 3.14
list("abcdefghijklmnopqrstuvwxyz")
> ['a', 'b', 'c', 'd', 'e', 'f', 'g', ...]
```

However, this isn't guaranteed

```
int("one") # ValueError
int("3.14") # ValueError
list(1) # TypeError
```

```
# However, str will alway succeed
# It'll provide a representation of the object
str({}) # Success
> "{}"
str(list)
> "<class 'list'>"
```

Note:

- Converting to str is always guareteed

Math Symbols

Name	Operator	Equivelent
Plus	`+`	1 + 2
Minus	`-`	1 - 2
Divide	`/`	1 ÷ 2
Multiply	`*`	1 x 2
Exponent	`**`	1^2

Note:

- Since ÷ was hard to reach on they keyboard, languages use / as divide
- Also since x is a letter, we use * instead
- If you've ever used excel, it uses the same symbols

```
2 * 3 + 4
>>> 9
2 * (3 + 4)
>>> 14

radius = 3
pi = 3.14
area = pi * (radius ** 2)
```

Note:

- Also BIDMAS order of operations from maths also applies

Operator	Name
==	Equals
!=	Not Equals
<	Less Than
<=	Less Than or Equals
>	Greater Than
>=	Greater Than or Equals

```
1 == 1
>>> True
5 > 10
>>> False
```

Watch out for `==`

Since `=` is used for assigning variables

```
a = 1
a == 1
> True
```

Note:

- Since `=` is used for creating variables

If Statements

Note:
If statements allow you to do more complex logic

```
area = 4
if area < 1:
    print("Area is less than 1")
```

The code runs `area < 1`

```
area = 4
if False:
    print("Area is less than 1")
```

Note:

- If statements allows you to optionally run code based on some condition

---

If statements use indents to decide what inside it

```
if True:
    print("This is run in the if statement")
    print("So is this")
    if True:
            print("This is another block")


print("This is outside the block")
```

You can also use `else` to run something if it's not true

```
a = 1
if a < 0:
    print("a is less than 1")
```

```
else:
        print("a is bigger than 1")
```

```
a = 1
if a == 1:
        print("A is 1")
elif a < 10:
    print("A is less than 10")
else:
        print("A is bigger than 10")
```

Note:
You can also run code

You can use `and` to require two condition to be true

You can use `or` to require either conditions to be true

```
student = True
healthy = True

if student and healthy:
    print("You're a healthy student")
elif student or healthy:
    print("You're either healthy or a student")
else:
        print("You're not healthy or a student")
```

For Loops

```
for x in range(3):
    print(x)

> 0
> 1
> 2
```

Range is just a shorthand for creating a list

Watch out range starts at 0

```
list(range(3)) == [0, 1, 2]
```

It's quite common to iterate over lists

```
fruits = ["apple", "pear", "orange"]
fruit_count = len(fruits)
for x in range(fruit_count):
    fruit = fruits[x]
    print(fruit)

> "apple"
> "pear"
> "orange"
```

Note:

For iterates over lists

```
fruits = ["apple", "pear", "orange"]
for fruit in fruits:
    print(fruit)

> "apple"
> "pear"
> "orange"
```

While Loops

```
logging_in = True
while logging_in:
    print("Please log in")

    username = input("Username: ")
        if username == "ben":
                logging_in = False
```

Loop Forever

```
while True:
    print("the end is never ")
```

Loop Control

You can use `continue` to repeat a loop

```
for x in range(10):
    is_even = x % 2 == 0
    if is_even:
            continue

        print("Odd", x)
```

```
while True:
    user_input = input("Type the letter '1'")
    if user_input != "1":
            continue

        print("user typed the letter '1'")
```

You can use `break` to exit a loop early

```
fruits = ["apple", "bannana", "cashew"]
for fruit in fruits:
        first_letter = fruit[0]
        if first_letter == "a":
                print(fruit, "begins with 'a'")
                break

> apple begins with 'a'
```

```
while True:
    user_input = input("Type q to exit")
    if user_input == "q":
        break
```

Input & Output

Print out values using print

```
print(1)
>>> 1
```

You can also print out multiple values

```
a = 1
b = 2
print(a, b)
```

Take user input using input

```
name = input("Enter Your Name:")
print(name)

>>> "Ben"
```

Functions

Note:

- In your code your going to have a lot repeating blocks

```
print("Hi there! :)")
```

runs the same as

```
def say_hello():
    print("Hi there! :)")

say_hello()
> "Hi There"
```

Note:
This allows you to group blocks of code together
On it's own this isn't very useful, however... NEXT SLIDE

You also pass variables into a function

These are called parameters or arguments

```
def say_hello(name):
    print("Hi there, ", name)

say_hello("Ben")
say_hello("Holly")
> "Hi there, Ben"
> "Hi there, Holly"
```

```
name = "Ben"
print("Hi there, ", name)
name = "Holly"
print("Hi there, ", name)
```

Parameters

```
def print_parameters(a, b):
    print(f"a: {a}")
    print(f"b: {b}")


print_parameters(1, "foo")
> a: 1
> b: foo
```

You can also return values back from a function

```
def calculate_rectangle_area(width, height):
    return width * height

def calculate_circle_area(radius):
        PI = 3.14
        return PI * (radius ** 2)


area = calculate_rectangle_area(5, 5)
print(area)
> 25
area = calculate_circle_area(5)
print(area)
> 78.5
```

You can also return nothing

This is useful for exiting a function early

```
logged_in = False
def show_account_info():
    if not logged_In:
        return

        print("Secret account info")
```

Function Tidbits

These aren't necessary or required ton know

But they're useful to know

Note:

Keyword Arguments

```
def calculate_area(width, height):
    return width * height

calculate_area(20, 10)
# is the same as
calculate_area(width=20, height=10)
# or even
calculate_area(
        height=10,
        width=20
)
```

Note:

- You can manual specify the parameters by name
- This means you can change the order

Type Hints

```
def calculate_area(width: int, height: int) -> int:
    return width * height

calculate_area(4, 4)
> 16
```

Type hints allow your IDE to warn you if your using the function wrong

Note:

- Type hints allow you indicate what types you want your parameters to be
- In this example, they hint that the LASER POINTER
    - width and height should be ints
    - returns an int

General Advice

File Handling

```
f = open("input.txt")
text = f.read()

print(text)
> "This is the contents of input.txt"

# Make sure your close a file or bad things happen
f.close()
```

Instead use a `with` statement

This automatically closes the file when you exit the statement, even if it errors

```
with open("input.txt") as f:
    text = f.read()

print(text)
> "This is the contents of input.txt"
```

You can write to files by opening them in write mode

```
# "w" specified the filemode, "w" means write
with open("output.txt", "w") as f:
    f.write("Hello!")
```

Errors

Sometimes functions can raise errors or exceptions

```
with open("file.txt") as f:
        text = f.read()
# FileNotFoundError: No such file or directory
```

You can also raise errors like this

```
raise ValueError("Invalid Value")
```

You can catch errors using try, except

```
try:
        with open("file.txt") as f:
                text = f.read()
        print(text)
except FileNotFoundError:
        print("File does not exist")
```

Note:

You can also catch mutliple errors

```
try:
        with open("file.txt") as f:
                text = f.read()
        print(text)
except FileNotFoundError:
    print("File does not exist")
except IsADirectoryError:
    print("File is a folder")
```

or group errors like this

```
try:
        with open("file.txt") as f:
                text = f.read()
        print(text)
```

```
except (FileNotFoundError, IsADirectoryError):
        print("File is invalid")
```

You can also catch all errors by not specifying an error

```
try:
    raise
except:
        pass
```

Imports

Note:

- Being able to import other files and packages is extremely important when working on any project

---

```
# square.py
def calculate_area(width: int, height: int) -> int:
        return width * height

def calculate_perimeter(width: int, height: int) -> int:
        return (width * 2) + (height * 2)
```

Note:

- Say we find our main file has gotten too big
- We can take some of the functions and drag them into their own file

---

```python
# app.py
import square

square.calculate_area(width=6, height=4)
> 8
```

Note:

- We can then import it into another file through an import
- The import name is the name of the file

```python
from square import calculate_area

calculate_area(width=6, height=4)
> 24
```

Note:

- We can also import individual variables by using a from import

---

```python
# modules/square.py
def calculate_area(width: int, height: int) -> int:
        return width * height
```

```python
# app.py
from modules import square
# or
from modules.square import calculate_area
```

The best ways to learn

The best way to learn is by making something

If you have a problem, look it up.

StackOverflow usually gives good answers

Use tutorial websites

W3Schools is really good

If you want to do some practice

Try some online coding questions like leetcode

Tasks

Code is available at

https://github.com/bu-compsecsoc/programming-crash-course

If you need any help, join our discord: bucss.net/discord

Note:

- There's loads of people willing to be able to provide programming help
- Also if your watching through Youtube, you can join as well
- If you'd like me to go anything again, I can