Links

Sunday, November 19, 2023 9:51 PM

GitHub Repository

- https://github.com/vivekduttamishra/202311-bosch-python02

This Notebook

- 202311-bosch-python02
- https://onedrive.live.com/?authkey=%21AEtnGCHNAnWzX7s&id=FE2CA4D12AD5D349%211150715&cid=FE2CA4D12AD5D349&parId=root&parQt=sharedby&parCid=UnAuth&action=defaultclick

Pretest

- https://docs.google.com/forms/d/e/1FAIpQLSdZfBXnGFHpfprs9_7yY_8nI_Cd9zYjrptRK4IYK3QuSTOLcw/viewform?usp=sf_link

Post your Questions

- https://docs.google.com/document/d/11uz-vHf8iq5f7Mg3DgDElumTeybTTICbTQih1PU2nWM/edit?usp=sharing

Hello World

Monday, November 20, 2023 10:45 AM

Hello World C

```
#include <stdio.h>

int main()
{
	printf("Hello World\n");
	return 0;
}
```

Hello World Java

```
class Program
{
	public static void main()
	{
		System.out.println("Hello World");

	}
}
```

Hello World Python

```
print('Hello World')
```

- Python is one of the simplest langauges.

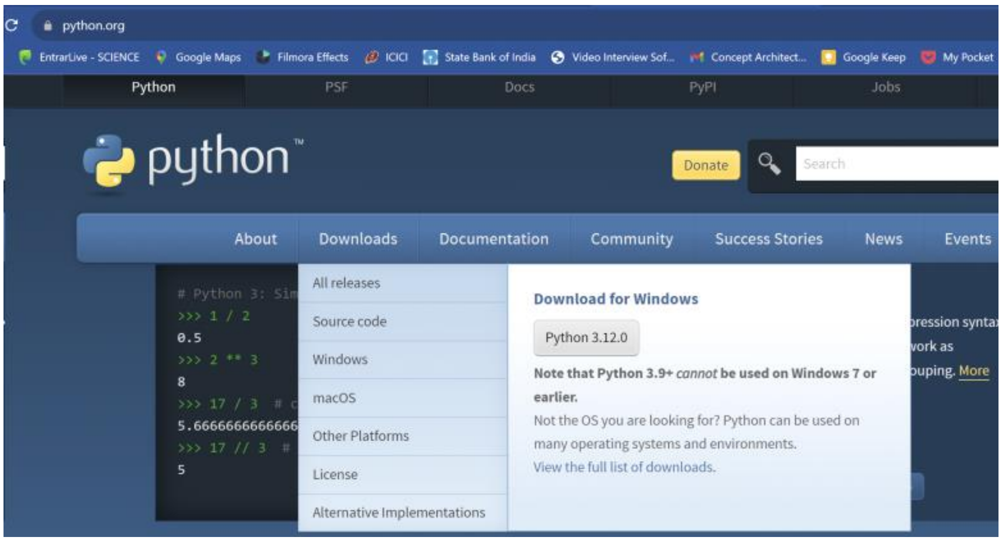
Python Enviornment

Monday, November 20, 2023 11:07 AM

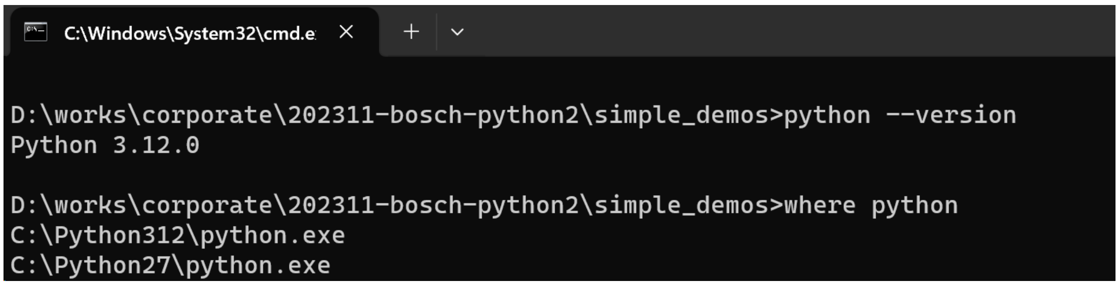
- python is an interpreted (scripting) language.
- You need a python interpreter to run the code.

Intall Python

https://python.org

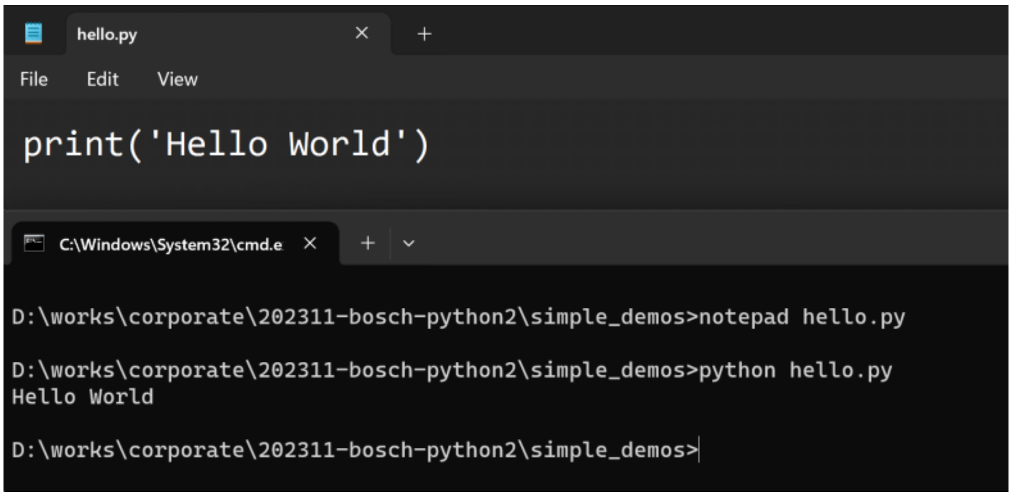

python.org
Python
PSF
Docs
PyPI
Jobs
python
Donate
Search
About
Downloads
Documentation
Community
Success Stories
News
Events
All releases
Source code
Windows
macOS
Other Platforms
License
Alternative Implementations
Download for Windows
Python 3.12.0
Note that Python 3.9+ cannot be used on Windows 7 or earlier.
Not the OS you are looking for? Python can be used on many operating systems and environments.
View the full list of downloads.

Validate Python



```
C:\Windows\System32\cmd.e

D:\works\corporate\202311-bosch-python2\simple_demos>python --version
Python 3.12.0

D:\works\corporate\202311-bosch-python2\simple_demos>where python
C:\Python312\python.exe
C:\Python27\python.exe
```

Hello World Python (Script)

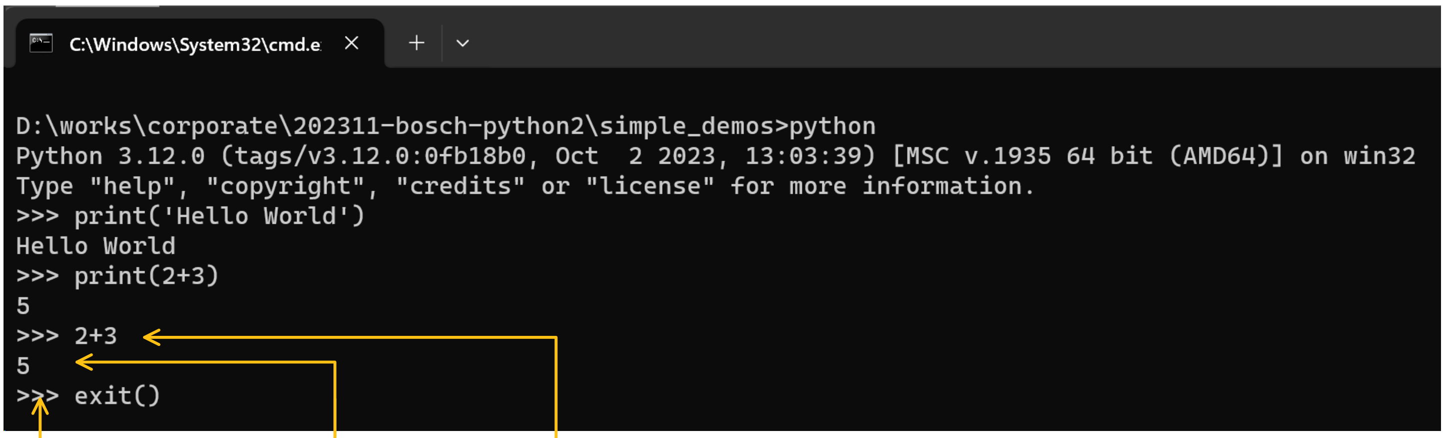


```
hello.py
File  Edit  View

print('Hello World')
```

```
C:\Windows\System32\cmd.e

D:\works\corporate\202311-bosch-python2\simple_demos>notepad hello.py

D:\works\corporate\202311-bosch-python2\simple_demos>python hello.py
Hello World

D:\works\corporate\202311-bosch-python2\simple_demos>
```

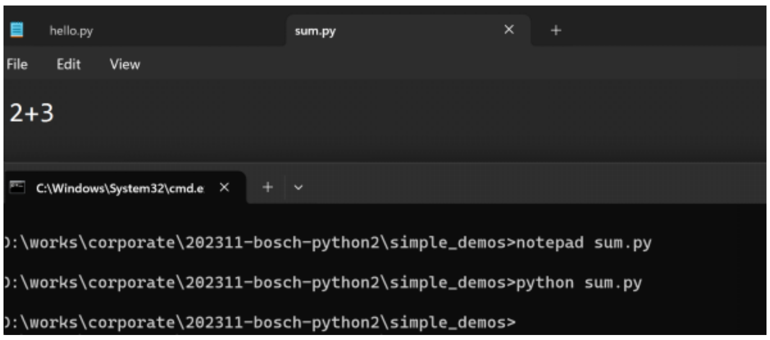
Python Shell aka REPL (Read Execute Print Loop)

- python code can be executed in an interactive shell envioronenment
- We can enter the environment by typing "python" without arguments
- We can exist the environment by calling exit()



```
D:\works\corporate\202311-bosch-python2\simple_demos>python
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello World')
Hello World
>>> print(2+3)
5
>>> 2+3
5
>>> exit()
```

- Read a line of python
- Execute it internally (2+3=5)
- Print the result
  - In Shell print is implicit
  - In script print will not happen unless you call print
- Loop back to get more user input



```
2+3
```

```
):\works\corporate\202311-bosch-python2\simple_demos>notepad sum.py

):\works\corporate\202311-bosch-python2\simple_demos>python sum.py

):\works\corporate\202311-bosch-python2\simple_demos>
```

IDE

Monday, November 20, 2023 11:43 AM

- Generally, we will be using some IDE to develop our application.

Common IDEs

1. IDLE (comes as part of standard Python installation)

- A hybrid between SHELL and editor
- can be used for smaller apps.
- NOT Recommended.

2. Eclipse/Visual Studio

- can be configured for Python applications.
- we can use if we are familiar with this IDE

3. pycharm

- one of the python specific IDE
- popular with Python developers.

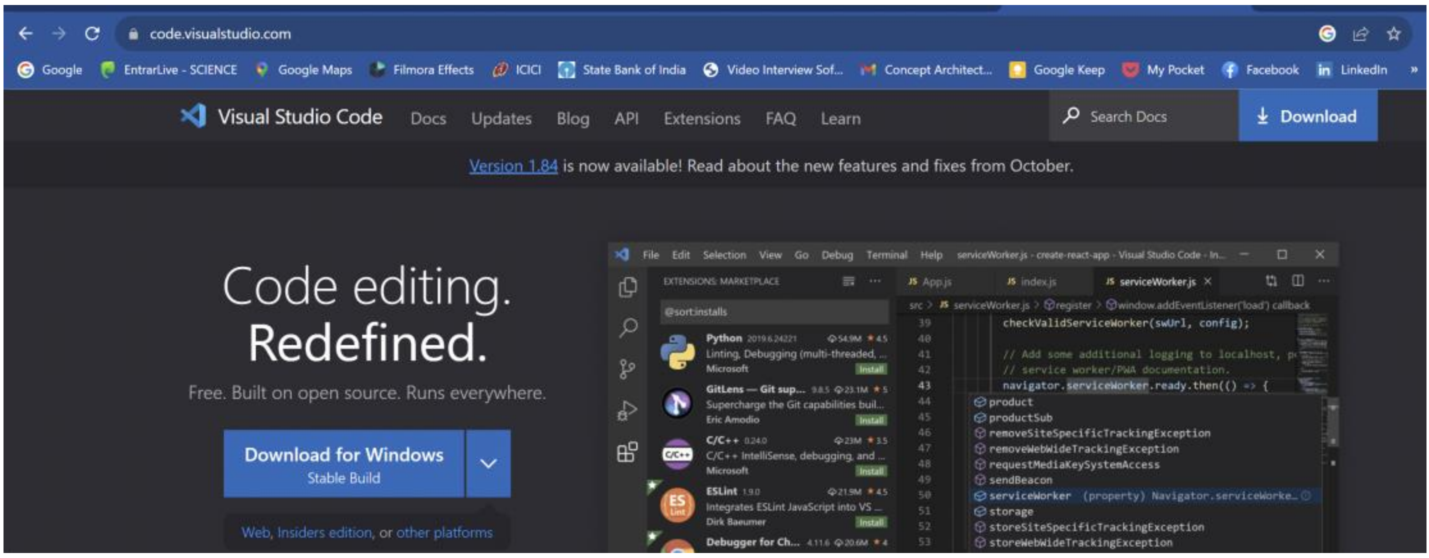
4. Visual Studio Code (VS Code)

- One of the most featurerich IDE
- Popular with most programming langauges today

https://code.visualstudio.com/

VS Code

Monday, November 20, 2023 11:56 AM

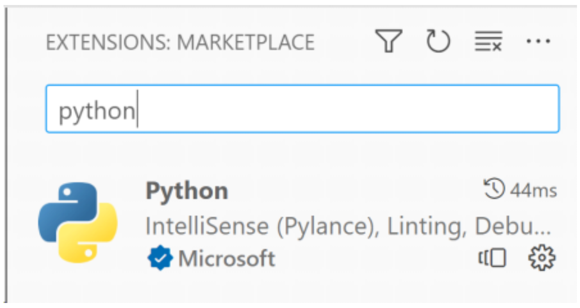


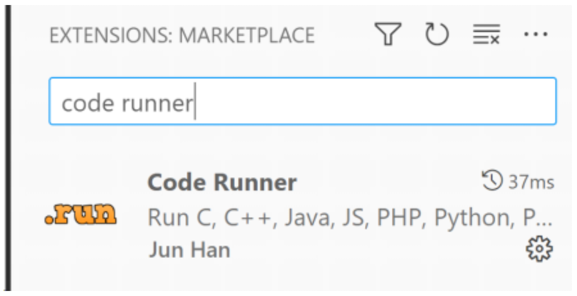
Version 1.84 is now available! Read about the new features and fixes from October.

Code editing.
Redefined.

Free. Built on open source. Runs everywhere.

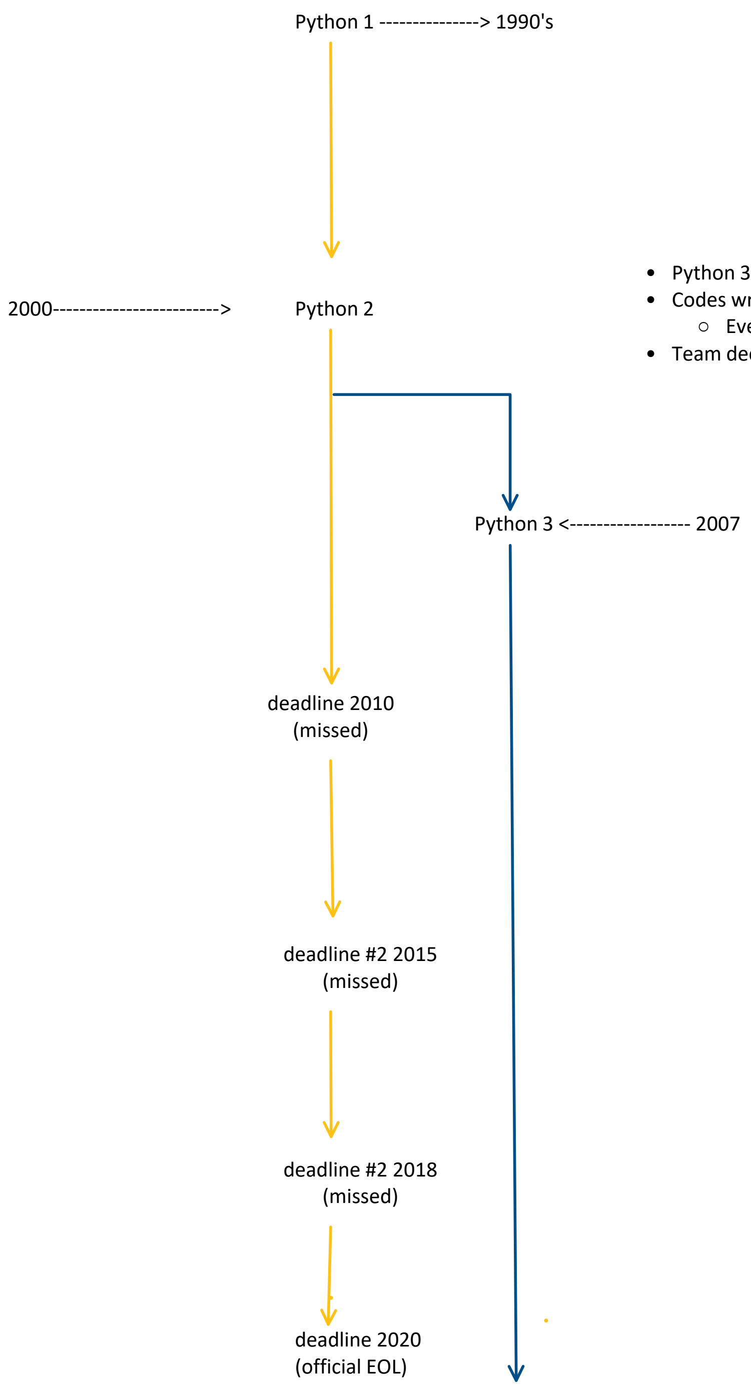
Prepare VS Code for Python Development





Python Evolution.

Monday, November 20, 2023 12:02 PM


Python 1 ---------------> 1990's
2000-------------------------->
Python 2
Python 3 <------------------ 2007
deadline 2010 (missed)
deadline #2 2015 (missed)
deadline #2 2018 (missed)
deadline 2020 (official EOL)

- Python 3 brought several breaking changes.
- Codes written for Python 2 may not work for Python 3
  - Even the Hello World doesn't work!
- Team decided to support python 2 for sometime to ensure smooth transition.

Jupyter Notebook

Monday, November 20, 2023 12:15 PM

Step #1 Install the jupyter notebook enviornment

```
pip install jupyterlab
```

From <https://jupyter.org/install>

Step #2 Install the support for Python for jupyter notebook

```
python -m pip install ipykernel
```

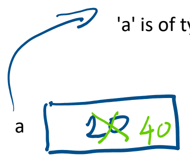
From <https://docs.posit.co/ide/server-pro/user/2023.03.1/jupyter-lab/guide/jupyter-kernel-management.html>

Variables are References

Monday, November 20, 2023 2:29 PM

C like language (C/C++/Java/C#)

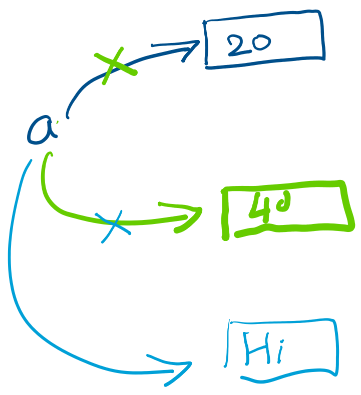
```
int a= 20;

a=40;

a="Hi" ; //NOT Allowed. a in 'int'
```



'a' is of type int that can store any int.

Python

```
a= 20

a=40

a="Hi"
```



- a doesn't have type or value
- It points to some memory location that has value 20
    - 20 is int.

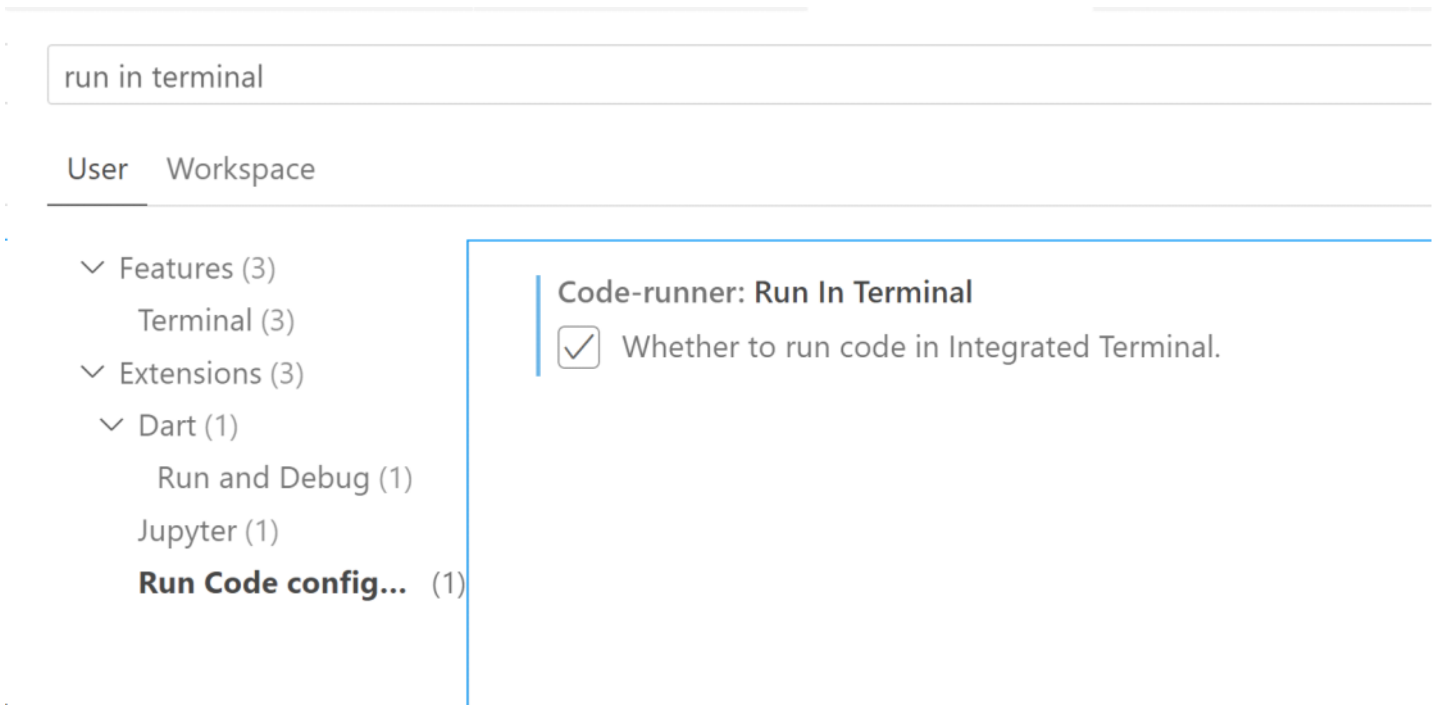
- now a refers to another memory that has value 40

- now a refers to "Hi"
- Now type(a) --> type("Hi") ---> str

VS Code configuration

Monday, November 20, 2023 2:51 PM

To run the code in the terminal and NOT in output window



run in terminal

User Workspace

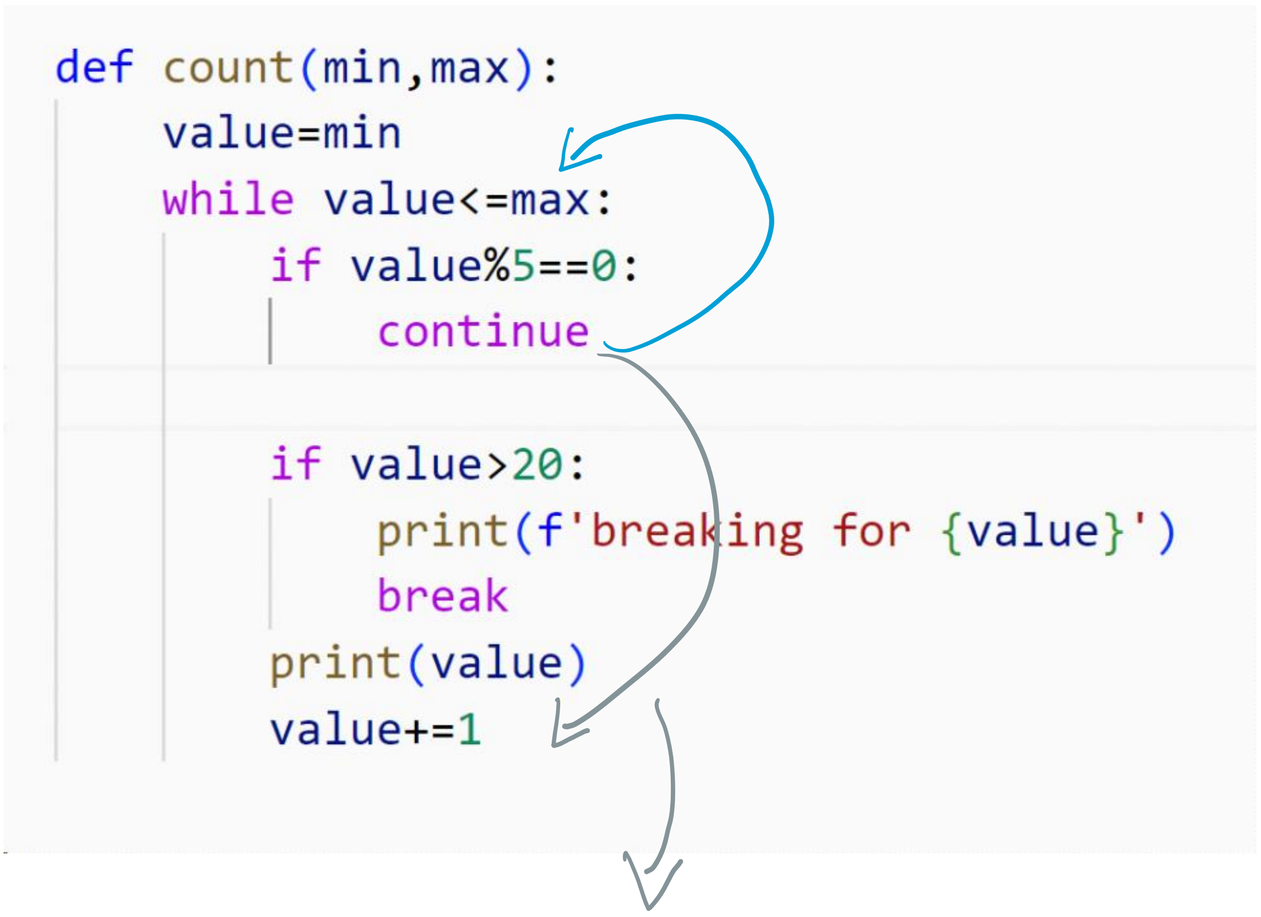
- Features (3)
  - Terminal (3)
- Extensions (3)
  - Dart (1)
    - Run and Debug (1)
  - Jupyter (1)
  - **Run Code config...** (1)

Code-runner: **Run In Terminal**

[✓] Whether to run code in Integrated Terminal.

Continue Problem

Monday, November 20, 2023 4:21 PM



```python
def count(min,max):
    value=min
    while value<=max:
        if value%5==0:
            continue

        if value>20:
            print(f'breaking for {value}')
            break
        print(value)
        value+=1
```

- skipped value increment
- it is now an endless loop

Assignment 1.1

Monday, November 20, 2023 4:24 PM

1. Write a program to take "n" numbers as user input and print the highest number

```
total? 5
number1? 20
number2? 12
number3? 35
number4? 14
number5? 5

Max is 35
```

2. Write a program to take "n" numbers as user input and print the second-highest number

```
total? 5
number1? 20
number2? 12
number3? 35
number4? 14
number5? 5

Second Highest is 20
```

3. Write a program to print the given pattern based on input "n"

```
n? 5

*
*    *
*    *      *
*    *      *    *
*    *      *    *      *
```

4. Write a program to check if a given number is prime or not

- A prime numebr is a number that is divisible only by 1 and itself.

Avoid Global codes

Tuesday, November 21, 2023 9:50 AM

1. python allows to write global codes (no need of a main() like function)
2. This makes a good use case as a beginners language (getting started)
3. But global codes have their inherent problem
    - They can overwrite each other.
4. We should avoid global codes as much as possible
    - It is not possible to remove global codes completely!

Functions and Scopes

- In Python a variable (reference) has two scopes
    - Local
        - those defined inside a function
    - Global
        - those defined outside a function.

Local Scope

- variables defined inside a function.
- they are created every time we call the function
- they are removed/destroyed when a function call is over.
- they are accessible only inside the function.
- we can have two different local variables with the same name in two different functions
    - they are treated differently and have no relationship.

Keywords vs standard library

Tuesday, November 21, 2023 10:53 AM

- Every language provides you two kind of predefined words out of box

Keywords

- these are language elements
    - example
        - statements like: if, for, while
        - other words like: global, return, break, continue
- these words are generally reserved.
- you can't create your own variable/reference/function with the same word.

```
for=20 # not allowed

if=30 # not allowed.
```

Standard Library

- these are predefined function or variables that comes as part of standard distribution (installation)
- They are helper functions/variables/classes that are provided out of box.
- But they are still not part of language keywords.
- They are just like user-defined functions created by a user in the Python developement team.
- These names are NOT reserved and can be overwritten
- Example
    - print, max, int, float,
        - Note unlike c style language int and float are not keywords or data type but convertor functions (not-reserved)

int=20 #allowed.

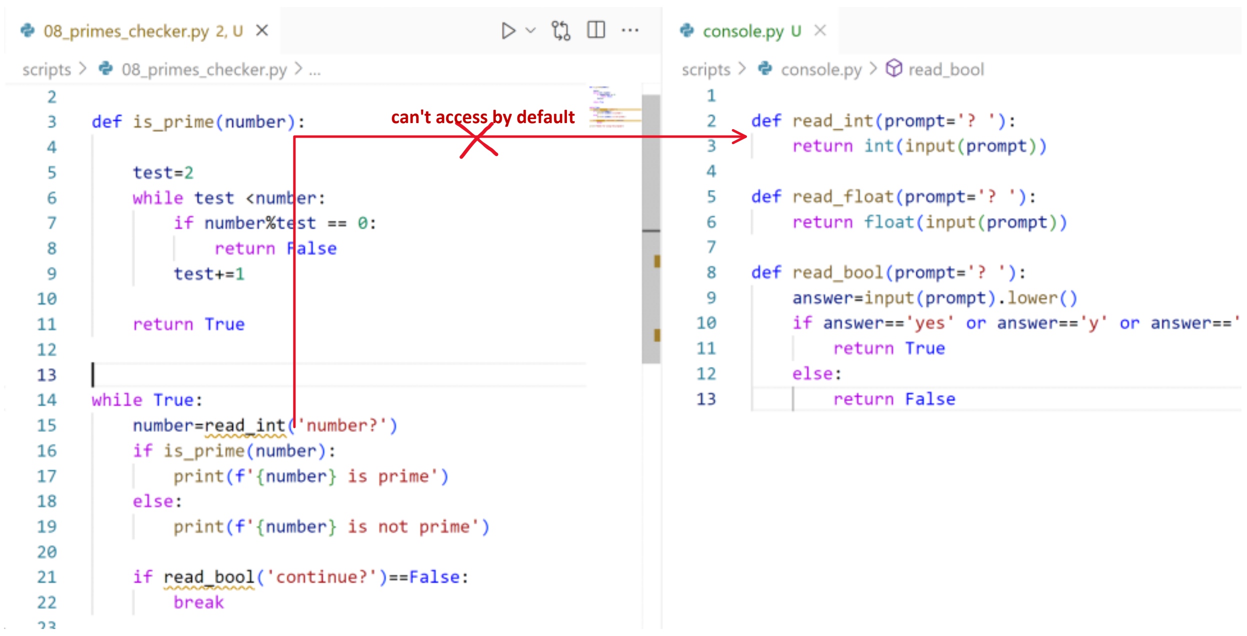
Reserved Words

- some languages may reserve extra words which are not keywords
- This may be done for future usage.
- python doesn't have any in this category.

Modules

Tuesday, November 21, 2023 11:05 AM

- By default, every python file is a module in itself.
- A module by default can access anything defined within it but not in another module.



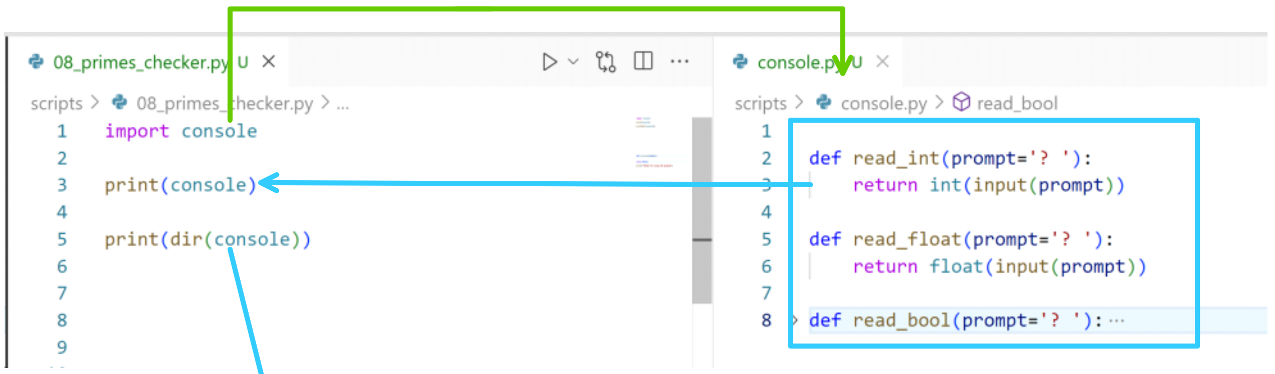
08_primes_checker.py 2, U
scripts > 08_primes_checker.py > ...
console.py U
scripts > console.py > read_bool
can't access by default

```python

def is_prime(number):

    test=2
    while test <number:
        if number%test == 0:
            return False
        test+=1

    return True

while True:
    number=read_int('number?')
    if is_prime(number):
        print(f'{number} is prime')
    else:
        print(f'{number} is not prime')

    if read_bool('continue?')==False:
        break
```

```python

def read_int(prompt='? '):
    return int(input(prompt))

def read_float(prompt='? '):
    return float(input(prompt))

def read_bool(prompt='? '):
    answer=input(prompt).lower()
    if answer=='yes' or answer=='y' or answer=='
        return True
    else:
        return False
```

To access that value we need to import "console" as module.

1. When we import a module, python searches for a .py file with the same name and executes it



08_primes_checker.py U
scripts > 08_primes_checker.py > ...
console.py U
scripts > console.py > read_bool

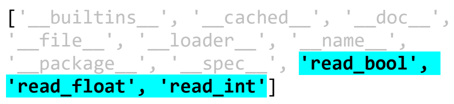
```python
import console

print(console)

print(dir(console))
```

```python

def read_int(prompt='? '):
    return int(input(prompt))

def read_float(prompt='? '):
    return float(input(prompt))

def read_bool(prompt='? '): ···
```

2. It creates a "module" object by bundelling all definitions in that file
    - the "console" becomes a reference or variable name in this case
    - IMPORTANT
        - python file name has become a variable name
        - module file name should follow same convention as we use for variable name
            - no blank space
            - no hyphen
            - no special char other than underscore
            - shouldn't be a reserved word like for.py or if.py
            - shouldn't begin with a digit.

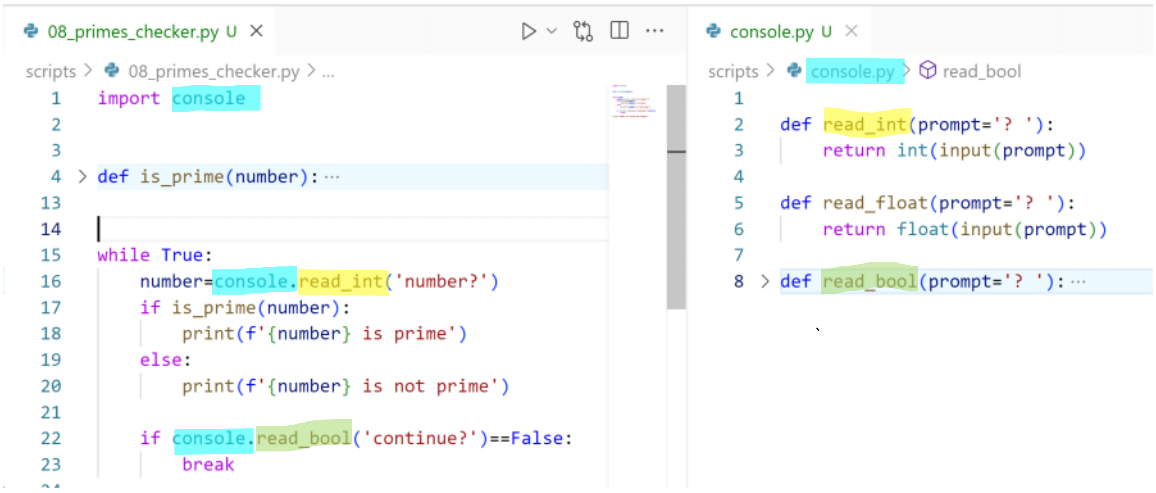
3. We can check an object's content using the "dir" command

- dir returns a list of elements present inside an object
- It applies not only to modules but any object in Python
    - every value in Python is an object.
    - even 'int' is an object
- here you may see some cryptic-looking names with double underscore prefixes and suffix.
    - we can ignore them for the time being.



```
['__builtins__', '__cached__', '__doc__',
'__file__', '__loader__', '__name__',
'__package__', '__spec__', 'read_bool',
'read_float', 'read_int']
```

How do we use module functions?

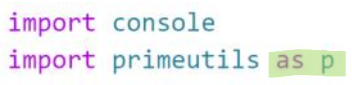

08_primes_checker.py U
scripts > 08_primes_checker.py > ...
console.py U
scripts > console.py > read_bool

```
import console


def is_prime(number): ···

while True:
    number=console.read_int('number?')
    if is_prime(number):
        print(f'{number} is prime')
    else:
        print(f'{number} is not prime')

    if console.read_bool('continue?')==False:
        break
```

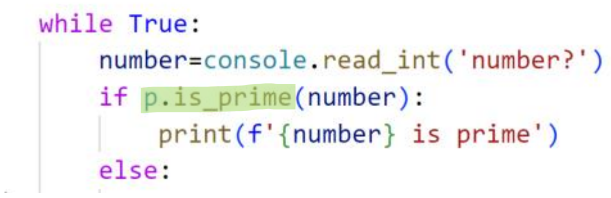
```

def read_int(prompt='? '):
    return int(input(prompt))

def read_float(prompt='? '):
    return float(input(prompt))

def read_bool(prompt='? '): ···
```

Module Alias

- if a module name is large we can shorten it by defining an alias name
- the module will be imported with an alias reference



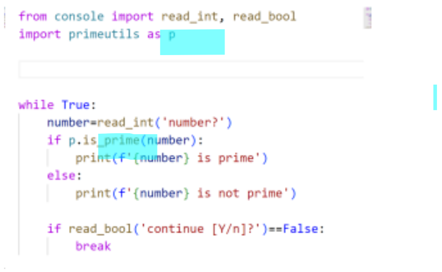
```
import console
import primeutils as p
```



```
while True:
    number=console.read_int('number?')
    if p.is_prime(number):
        print(f'{number} is prime')
    else:
```

Importing selected names from Module

- sometimes we need a specific element from the module
- We don't need all elements from the module.
- We may import specific element(s) directly in my module
    - now these elements will act as if they are defined in current module where they are imported.
    - they will act like global names
    - they don't need dot operator to use them
- Warining
    - this may ovewrite other global names.



```
from console import read_int, read_bool
import primeutils as p


while True:
    number=read_int('number?')
    if p.is_prime(number):
        print(f'{number} is prime')
    else:
        print(f'{number} is not prime')

    if read_bool('continue [Y/n]?')==False:
        break
```

Import all elements from a module

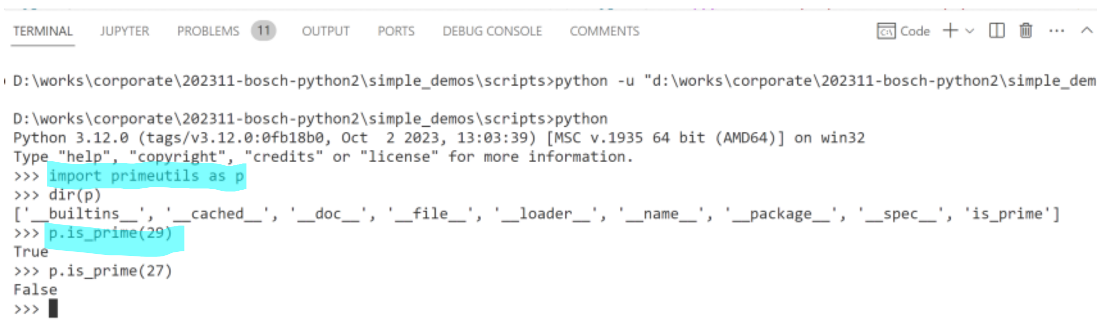
```
from console import *
```

- while this is allowed it is NOT RECOMMENDED.
- if we import all modules this way then we will have the same global problem we are trying to avoid.

- we don't know all the names we are getting from the module.

Modules can be imported by shell/repl/jupyter notebook

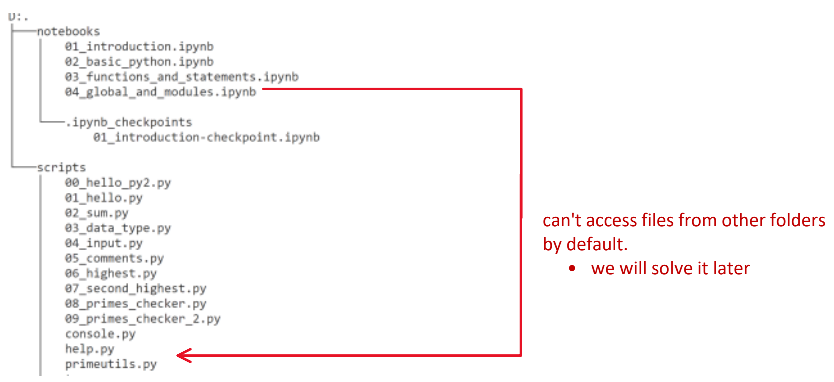
- while shell can define a module, it can import a module.



```
D:\works\corporate\202311-bosch-python2\simple_demos\scripts>python -u "d:\works\corporate\202311-bosch-python2\simple_dem

D:\works\corporate\202311-bosch-python2\simple_demos\scripts>python
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import primeutils as p
>>> dir(p)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'is_prime']
>>> p.is_prime(29)
True
>>> p.is_prime(27)
False
>>>
```

How is a module located.

- By default python searches for the module in the current working directory
- Since my primeutils.py file is present in scripts folder we can't access it in jupyter folder



```
D:.
├───notebooks
│       01_introduction.ipynb
│       02_basic_python.ipynb
│       03_functions_and_statements.ipynb
│       04_global_and_modules.ipynb
│
│   └───.ipynb_checkpoints
│           01_introduction-checkpoint.ipynb
│
└───scripts
        00_hello_py2.py
        01_hello.py
        02_sum.py
        03_data_type.py
        04_input.py
        05_comments.py
        06_highest.py
        07_second_highest.py
        08_primes_checker.py
        09_primes_checker_2.py
        console.py
        help.py
        primeutils.py
```

Important Standandard

Tuesday, November 21, 2023 11:45 AM

Assignment 2.1

Tuesday, November 21, 2023 12:02 PM

1. Write a program to print the given pattern based on input "n"

   n? 5

```
            *
          *   *
       *   *     *
     *   *     *   *
  *   *     *   *     *
```

HINT: python string has a "centre" function

- use help() not google()

Frequency

Tuesday, November 21, 2023 4:38 PM

Common Scenario

- you typically have a large set of values (Say 10000) with few unque keys
    - Example
        - month-wise sales report
            - we may have sold 500000 items in the last 12 months
            - there will be 500000 items grouped in 12 keys
- We will have large input list that will produce a short table with fewer keys.

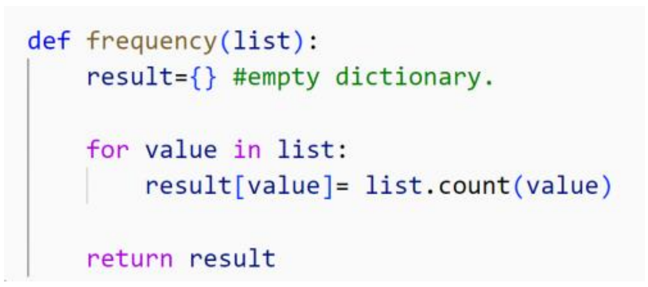
Imagine what count() do internally?

```
def count( seq, value):
    total=0
    for item in seq:
        if value==item:
            total+=1
    return total
```

- each call of count loops through entire list to find out the count of current item

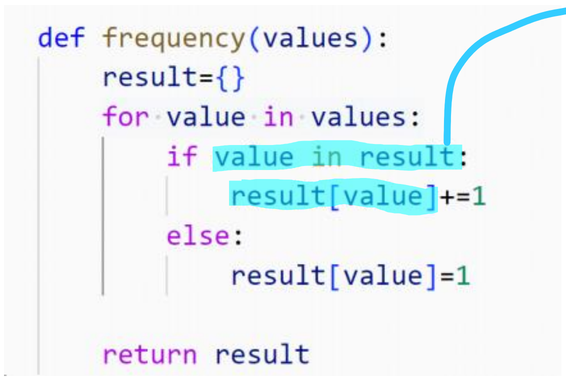
```
values= [2,3,2,3,2,2,3,5,2,2,5,2,... (5Lac)]

x= frequency(values)


 1. we start with the first item (value=2)
     • we count all occurrences of 2 using the count
       function
         ▪ it loops through all 5 lac items to count.

 2. then we move to second item (value=3)
     • we count all occurance of 3 using count function
         ▪ it loops through 5 lack items again
         ▪ It is acceptable for a different item.

 3. we take the third item (value=2) <--- again.
     • we already have calcuate the count in step 1
     • we recalcualte this couunt for same value to get
       same result
         ▪ But we will loop through all the 5 lack
           items again.


...
  • Total loop count for the operation is
     • 5lac * 5 lac
     • O(n^2)
```

Frequency V1



```
def frequency(list):
    result={} #empty dictionary.

    for value in list:
        result[value]= list.count(value)

    return result
```

Better Version



```
def frequency(values):
    result={}
    for value in values:
        if value in result:
            result[value]+=1
        else:
            result[value]=1

    return result
```

- there must be some loop here when we call in to check if this value is present in the dictionary or not
    - Why are we not including this in our complexity

Reason #1

- set and dict uses hashing algorithm which has typical compexity of O(1)
    - It doesn't use loops
    - It stores values in such a way that you can access it in minimum possible time.
        - It is NOT a linear search

- even if this were a linear model here we are searching in a very small list of 10-12 items
    - so the complexity will be negligible.

Assignment 2.2

Tuesday, November 21, 2023 5:29 PM

- write a function to print the calendar of a given month

example

print_calendar(1,1948)

- You are NOT supposed to use any builtin or third party function.

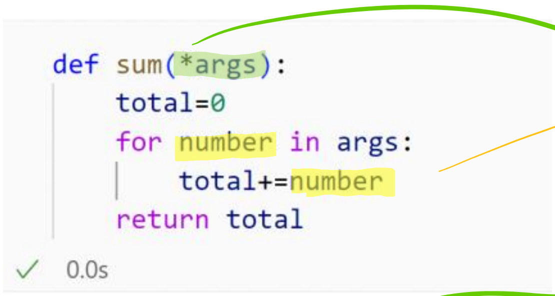
- Hint
    - You need to know
        - how many days are there in that month
            - 28
            - 29
            - 30
            - 31
        - what was the first day of the month

- You need to write another function that can return a week day for a given date.
    - exmple
        - what was the week day on 01/01/1948
    - Hint
        - 1/jan/1900 was a Monday
            - Can we find what day corresponds any given dd/mm/yyyy based on above information.

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Args

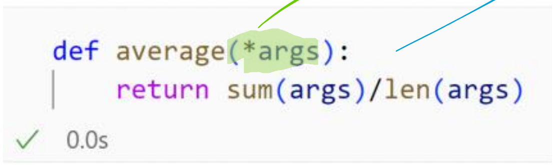
Wednesday, November 22, 2023 11:05 AM



```
def sum(*args):
    total=0
    for number in args:
        total+=number
    return total
```
✓ 0.0s

- here number is not a number but the inner tuple which is first (and only) item in the outer tuple.

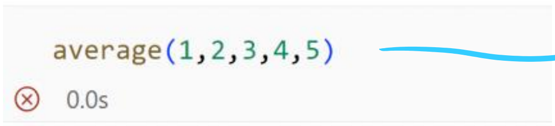
((1,2,3,4,5))



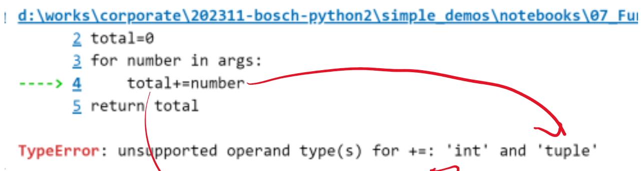
```
def average(*args):
    return sum(args)/len(args)
```
✓ 0.0s

(1,2,3,4,5)

*args wraps my values in a tuple



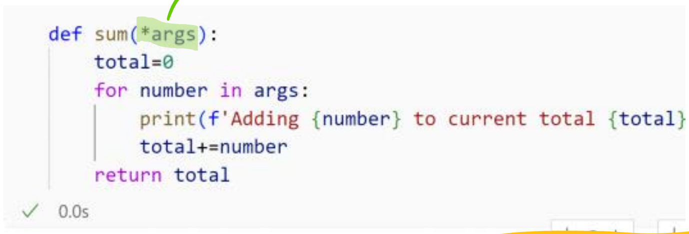
```
average(1,2,3,4,5)
```
ⓧ 0.0s



```
d:\works\corporate\202311-bosch-python2\simple_demos\notebooks\07_Fu
      2 total=0
      3 for number in args:
----> 4     total+=number
      5 return total

TypeError: unsupported operand type(s) for +=: 'int' and 'tuple'
```

Args unpacking Syntax.

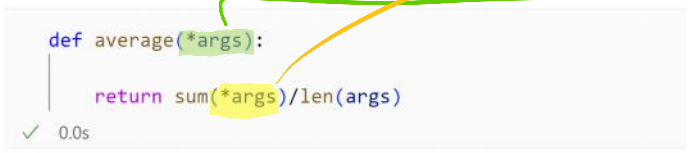
- We need to Unpack the tuple values in average before passing it to sum
- This syntax is often known as
    - unpacking syntax
    - spread syntax.
- This is also represented by "*" operator



```
def sum(*args):
    total=0
    for number in args:
        print(f'Adding {number} to current total {total}
        total+=number
    return total
```
✓ 0.0s

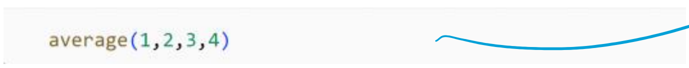
(~~(~~ 1, 2, 3, 4 ~~)~~)

- spread syntax spreads the value of a tuple as comma separated values



```
def average(*args):

    return sum(*args)/len(args)
```
✓ 0.0s

(1,2,3,4)



```
average(1,2,3,4)
```

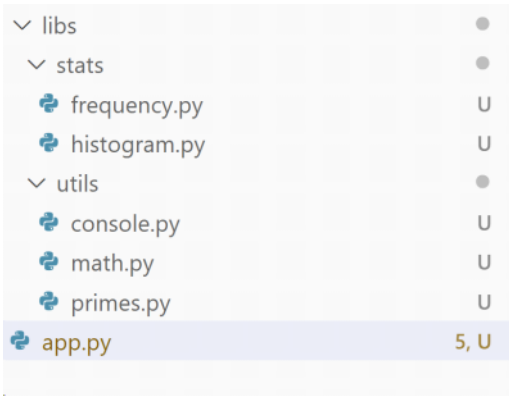
How do I know if a "*" means params or spread

- "*" means params only in function parameter declration.
- every where else (during function call) it is spread.

Modules Part 2

Wednesday, November 22, 2023 2:05 PM

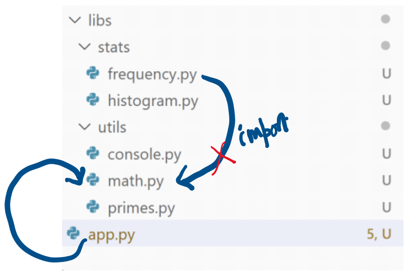
Consider a larger application with modules organized as sub modules



- we can access modules and sub-modules as dotted notation
- The modules are by default searched in current working directory

```
from libs.utils.console import read_int
import libs.utils.primes as p
import libs.stats.frequency as f
#from libs.stats.histogram import * # NOT RECOMMENDED. PREFER NEXT LINE
from libs.stats.histogram import plot_histogram
```

How do I access one module inside another



- Here frequency module needs functionality from math module
  - math module is not present under stats
  - both are present in two different sub directories.
- to locate math.py, we need
  - to go to parent folder (libs)
    - sub folder (utils)
      - locate math.py
- In python a module is located as absolute path starting from the location of main script.
  - whichever script is called at the begining.

Where should I keep my function check_args

- currently, it is present in the math module
- but it is not a mathematical function
- can you suggest a better module name?

Can I keep this function in the utils module?

Challenge?

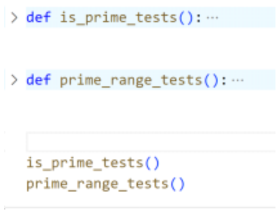
- utils is a folder
- you can't write Python code in a folder
- you will write in some file
- how do I put a function directly in folder module

__init__.py file

- a folder module can contain a python file with a special name __init__.py
- This file historically (till version 2) was needed in every module folder
  - Generally this file would be empty
  - if the file is not found current folder will not be treated as file.
  - This required is NOT deprecated in python 3+
- If this file is present (even in python3) any function or element defined in the file will be treated as defined the folder module.
  - we don't need to specify this name in the import statement.

```
from libs.utils import check_args
```

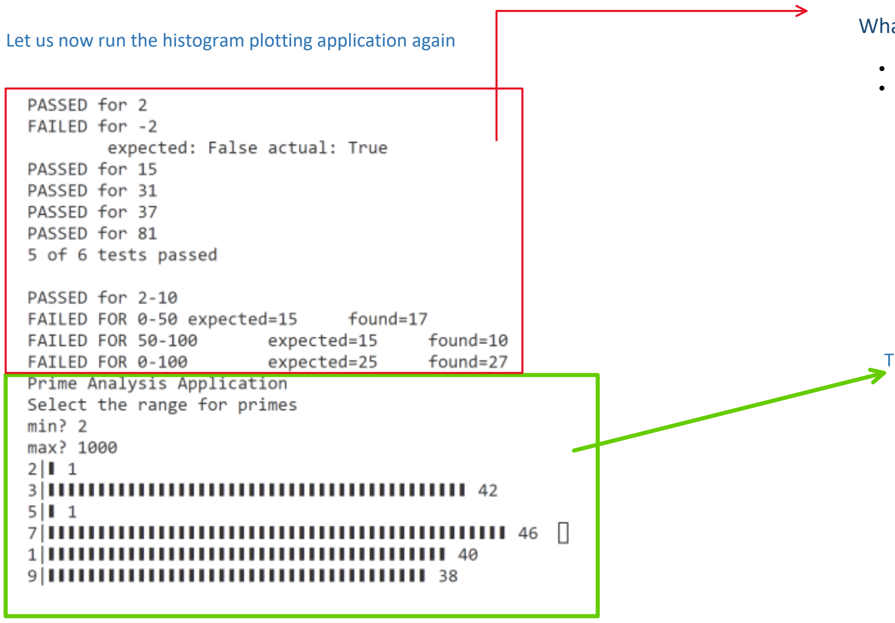
Let us write some test logic for our prime number function in primes.py file



```
> def is_prime_tests(): ...


> def prime_range_tests(): ...


is_prime_tests()
prime_range_tests()
```

- Here we have written some test code to test and verify if my prime logic is working
- It is a good idea to test your code properly

Let us now run the histogram plotting application again



```
PASSED for 2
FAILED for -2
        expected: False actual: True
PASSED for 15
PASSED for 31
PASSED for 37
PASSED for 81
5 of 6 tests passed

PASSED for 2-10
FAILED FOR 0-50 expected=15     found=17
FAILED FOR 50-100       expected=15     found=10
FAILED FOR 0-100        expected=25     found=27
Prime Analysis Application
Select the range for primes
min? 2
max? 1000
2|▌ 1
3|▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌ 42
5|▌ 1
7|▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌ 46  ▯
1|▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌ 40
9|▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌ 38
```

What is this code

- this is a code we wrote to test our module's logic
- While this is useful for module developer this is an undesriable output for the production application which doesn't need proof of correctness in the production output.

This is where my Application Really Starts

- everything printed uptil this point in time is an undesirable output

Why did this code execute?

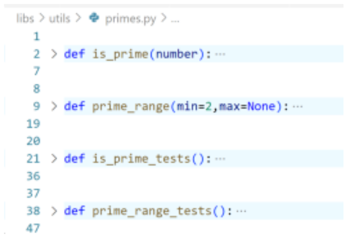
- Whenever we import a module (using any syntax) the module file is executed entirely by python runtime.
  - Even if we need only one function from the file, the whole file is still executed to generate that function.
  - Any global code written in that file will be executed.

How can we prevent execution of this unwanted code during module import.

- There are two approach

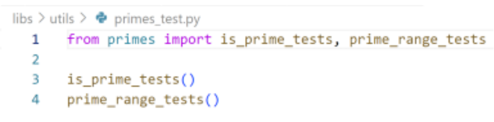
1. Generic Best Practice (Applies to all programming language) RECOMMENDED

- Avoid global codes.
- They have unwanted behavior
- Always write your tests in a separate file.



libs > utils > primes.py > ...

```
> def is_prime(number): ...


> def prime_range(min=2,max=None): ...


> def is_prime_tests(): ...


> def prime_range_tests(): ...
```



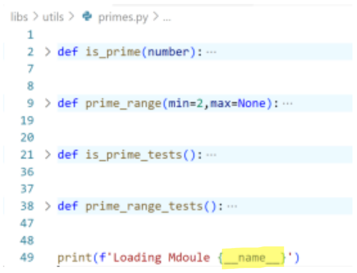
libs > utils > primes_test.py

```
from primes import is_prime_tests, prime_range_tests

is_prime_tests()
prime_range_tests()
```

2. Pythonic Way (POPULAR WAY WITH PYTHON DEVELOPERS)

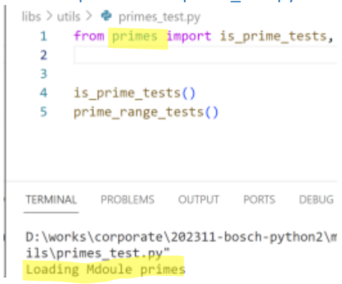
Every python module has a special property called __name__

- whenever you import a module the __name__ property will be set based on import path.



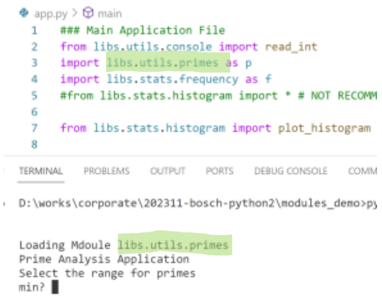
```
libs > utils > primes.py > ...

> def is_prime(number): ···


> def prime_range(min=2,max=None): ···


> def is_prime_tests(): ···


> def prime_range_tests(): ···


  print(f'Loading Mdoule {__name__}')
```

When Imported from prime_test.py



```
libs > utils > primes_test.py
from primes import is_prime_tests,


is_prime_tests()
prime_range_tests()
```

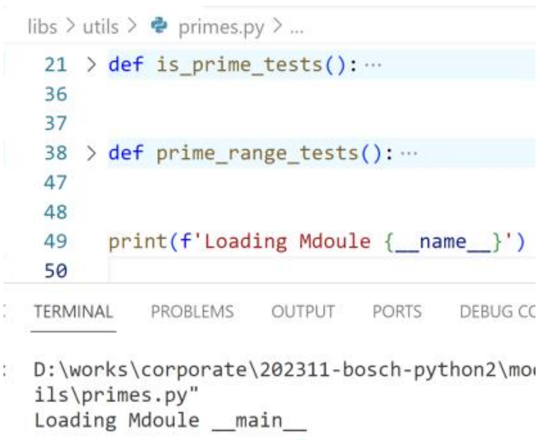
```
TERMINAL   PROBLEMS   OUTPUT   PORTS   DEBUG

D:\works\corporate\202311-bosch-python2\m
ils\primes_test.py"
Loading Mdoule primes
```



```
app.py > main
### Main Application File
from libs.utils.console import read_int
import libs.utils.primes as p
import libs.stats.frequency as f
#from libs.stats.histogram import * # NOT RECOMM

from libs.stats.histogram import plot_histogram
```

```
TERMINAL   PROBLEMS   OUTPUT   PORTS   DEBUG CONSOLE   COMM

D:\works\corporate\202311-bosch-python2\modules_demo>py

Loading Mdoule libs.utils.primes
Prime Analysis Application
Select the range for primes
min? █
```

But when a module is directly executed by python (NOT IMPORTED) it will always have same name __main__

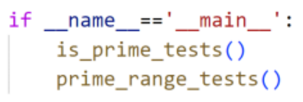


```
libs > utils > primes.py > ...
> def is_prime_tests(): ···


> def prime_range_tests(): ···


  print(f'Loading Mdoule {__name__}')

```

```
TERMINAL   PROBLEMS   OUTPUT   PORTS   DEBUG CC

D:\works\corporate\202311-bosch-python2\mo
ils\primes.py"
Loading Mdoule __main__
```

How does this logic Help?

- This can help us differentiate between situation
  - File is executed as an application ---> '__name__'=='__main__'
  - File is imported as a Module ---> __name__!='__main__'



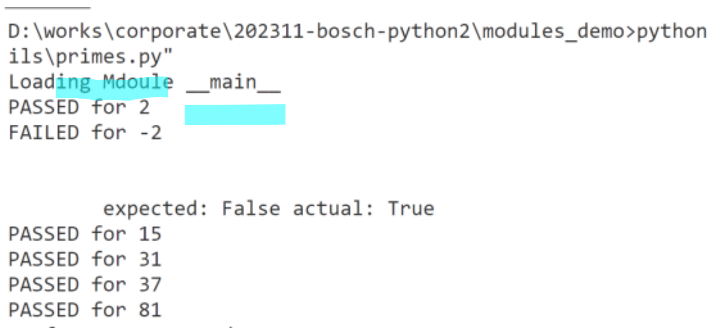
```
if __name__=='__main__':
    is_prime_tests()
    prime_range_tests()
```

What will Now Happen?

This test code will run if I ever try

$ python primes.py

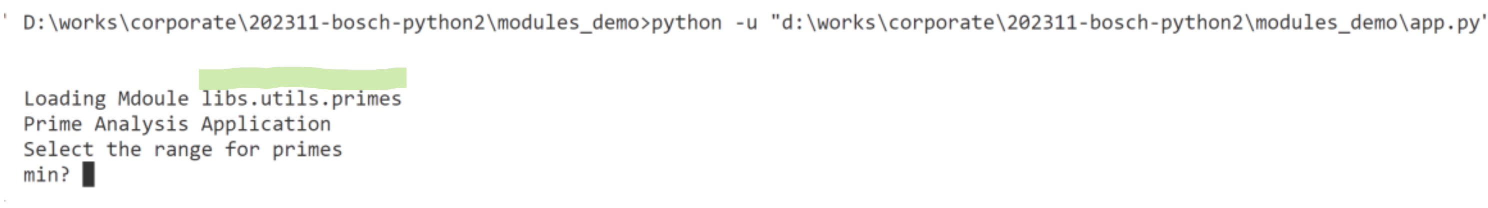
- In this case primes.py becomes the __main__ module



```
D:\works\corporate\202311-bosch-python2\modules_demo>python
ils\primes.py"
Loading Mdoule __main__
PASSED for 2
FAILED for -2


        expected: False actual: True
PASSED for 15
PASSED for 31
PASSED for 37
PASSED for 81
```

Test Test Code will NOT run if we import the module

- In such cases the name will be something other than __main__
- We don't enter the test block



```
D:\works\corporate\202311-bosch-python2\modules_demo>python -u "d:\works\corporate\202311-bosch-python2\modules_demo\app.py'


Loading Mdoule libs.utils.primes
Prime Analysis Application
Select the range for primes
min? █
```

How to Locate a Module

Wednesday, November 22, 2023 3:06 PM

- By default A module is searched relative to same directory from where the main module is loaded.
- This is useful when we create an application and want to load modules related to the application
- But sometimes we may have modules which are more generic and can be used with different projects.
- We don't want to store the modules under the project folder.

Common Module Location

- [common_mdoules_home]
  - [utils]
    - __init__.py
    - math.py
    - primes.py
    - console.py
  - [stats]
    - histogram.py
    - frequency.py

Let us assume we are working on multiple projects which are present in different folders

- [ecommerce_project]
  - app.py
  - [products]
    - product.py
    - product_manager.py
  - [orders]
    - order.py
    - order_manager.py
  - [users]
    - user.py
    - user_manager.py
    - user_analsys.py

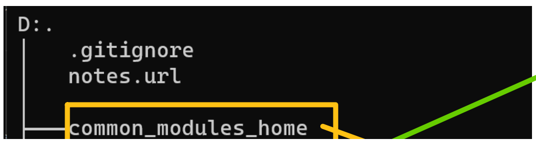
- [ems_project]
  - app.py
  - [employees]
    - employee.py
    - employee_manager.py
  - [deptartment]
    - department.py
    - department_manager.py

- Here we have two separate projects present in their respective folders
- Each project has project specific modules
- But both project may still need common modules like
  - histogram()
  - read_int()
- But the common modules are not present in their workspace folder.

How python searches for Modules

- A python runtime maintains a list of paths from where it searches for All the modules including
  - pre-installed modules
  - third-party modules
  - user-defined modules

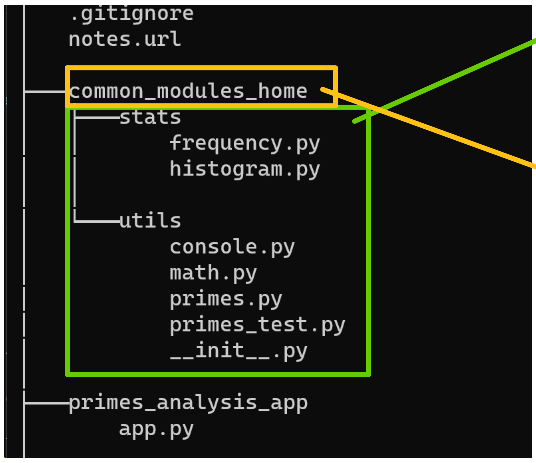
- It populates this list from these sources
  1. It knows the installation directory of python runtime where standard modules are present
  2. It knows the location of directory where third party modules are downloaded by PIP command
  3. It always searches in the working directory from where __main__ module is loaded.
  4. We can provide more locations using a special OS level environment variabled called PYTHONPATH
  5. The rule of creating this variable depends on OS and NOT on python
     - You will create it the same way you created PATH variable in the OS.



```
D:.
│   .gitignore
│   notes.url
│
├───common_modules_home
```

Common Reusable Modules

- Not present in the application folder
- By default app.py can not locate these modules.



```
    .gitignore
    notes.url

├───common_modules_home
│   ├───stats
│   │       frequency.py
│   │       histogram.py
│   │
│   └───utils
│           console.py
│           math.py
│           primes.py
│           primes_test.py
│           __init__.py
│
├───primes_analysis_app
│       app.py
```

Common Reusable Modules

- Not present in the application folder
- By default app.py can not locate these modules.

Add This path to PYTHONPATH

- We don't add module folders to PYTHONPATH
- We add the parent folder of modules in PYTHONPATH

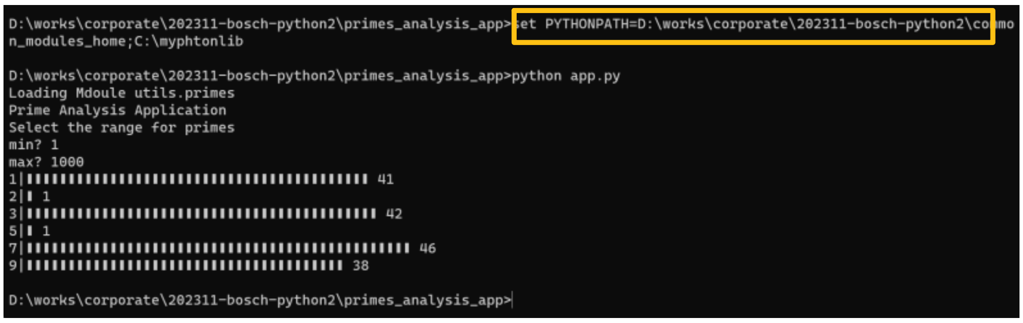
Why is common_modules_home not a module folder but parent of module folder?

- How do we know common_modules_home is not a module but its parent folder
- Why is utils and stats not parent folder
- This is my descision. period.
  - module folder will appear in python source code in import statement
  - the parent will appear in PYTHONPATH but NOT in import statement

- if we decide to make python_module_home as a module our import statements will change
  - import common_modules_home.utils.primes as p

- if we decide to add utils in PYTHONPATH, import will again change
  - import primes as p

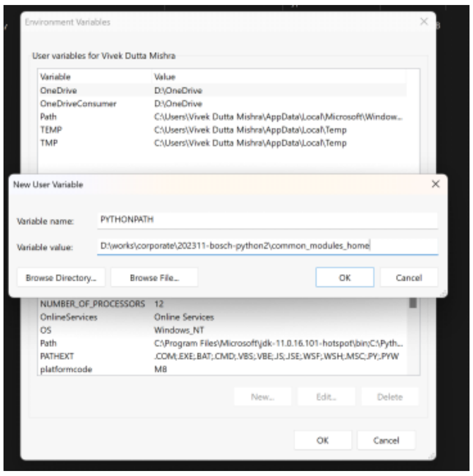
Setting PYTHONPATH at command window



```
D:\works\corporate\202311-bosch-python2\primes_analysis_app>set PYTHONPATH=D:\works\corporate\202311-bosch-python2\commo
n_modules_home;C:\myphtonlib

D:\works\corporate\202311-bosch-python2\primes_analysis_app>python app.py
Loading Mdoule utils.primes
Prime Analysis Application
Select the range for primes
min? 1
max? 1000
1|▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮ 41
2|▮ 1
3|▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮ 42
5|▮ 1
7|▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮ 46
9|▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮ 38

D:\works\corporate\202311-bosch-python2\primes_analysis_app>
```

- This setting will be active only as long as terminal is active
- Once terminal is closed this is removed.
- This will not be available on any ohter terminal

Setting PYTHON PATH is system enviorment setting



Environment Variables

User variables for Vivek Dutta Mishra

Variable	Value
OneDrive	D:\OneDrive
OneDriveConsumer	D:\OneDrive
Path	C:\Users\Vivek Dutta Mishra\AppData\Local\Microsoft\Window...
TEMP	C:\Users\Vivek Dutta Mishra\AppData\Local\Temp
TMP	C:\Users\Vivek Dutta Mishra\AppData\Local\Temp

New User Variable

Variable name: PYTHONPATH

Variable value: D:\works\corporate\202311-bosch-python2\common_modules_home

NUMBER_OF_PROCESSORS	12
OnlineServices	Online Services
OS	Windows_NT
Path	C:\Program Files\Microsoft\jdk-11.0.16.101-hotspot\bin;C:\Pyth...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW
platformcode	M8

- IMPORTANT

    - Restart your shell/IDE/Jupyter for the change to take effect.
    - We may not add any folder in global PYTHONPATH unless it is useful to multiple application.

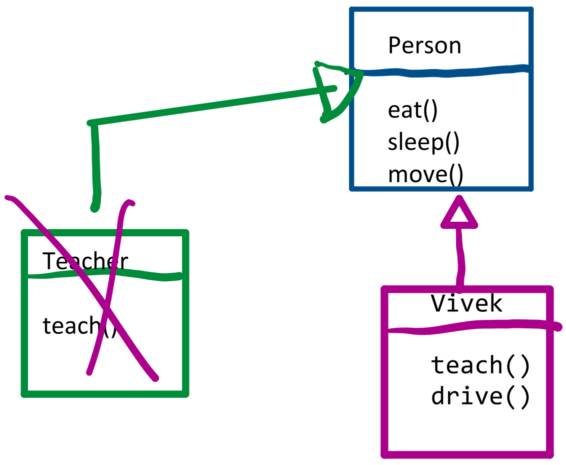
Internal Management of module directories.

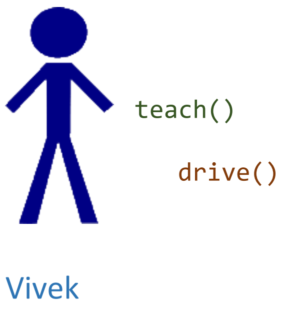
- python maintains a directory list where Python modules are searched.
- This list automatically includes
    - python installation directory
    - current working directory
    - all directories mentioned in PYTHONPATH

- this list is present in sys module as "path" variable

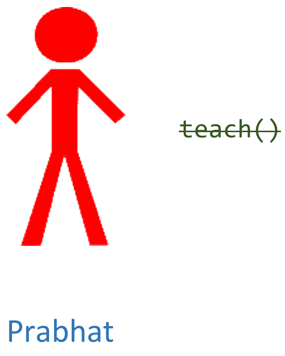
Object Model

Wednesday, November 22, 2023 4:35 PM





- Vivek aquired several behaviors after birth (creation of object)
- how can these properteis or behavior be part of Vivek's class



```
~~Person vivek~~

~~Teacher vivek~~

Vivek vivek

vivek.eat()
vivek.sleep()
```

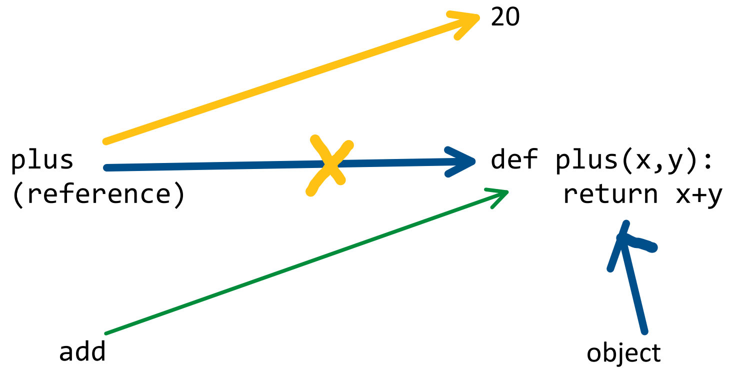
```
Person prabhat

prabhat.eat()
prabhat.sleep()
```

Function Internals

Thursday, November 23, 2023 10:48 AM

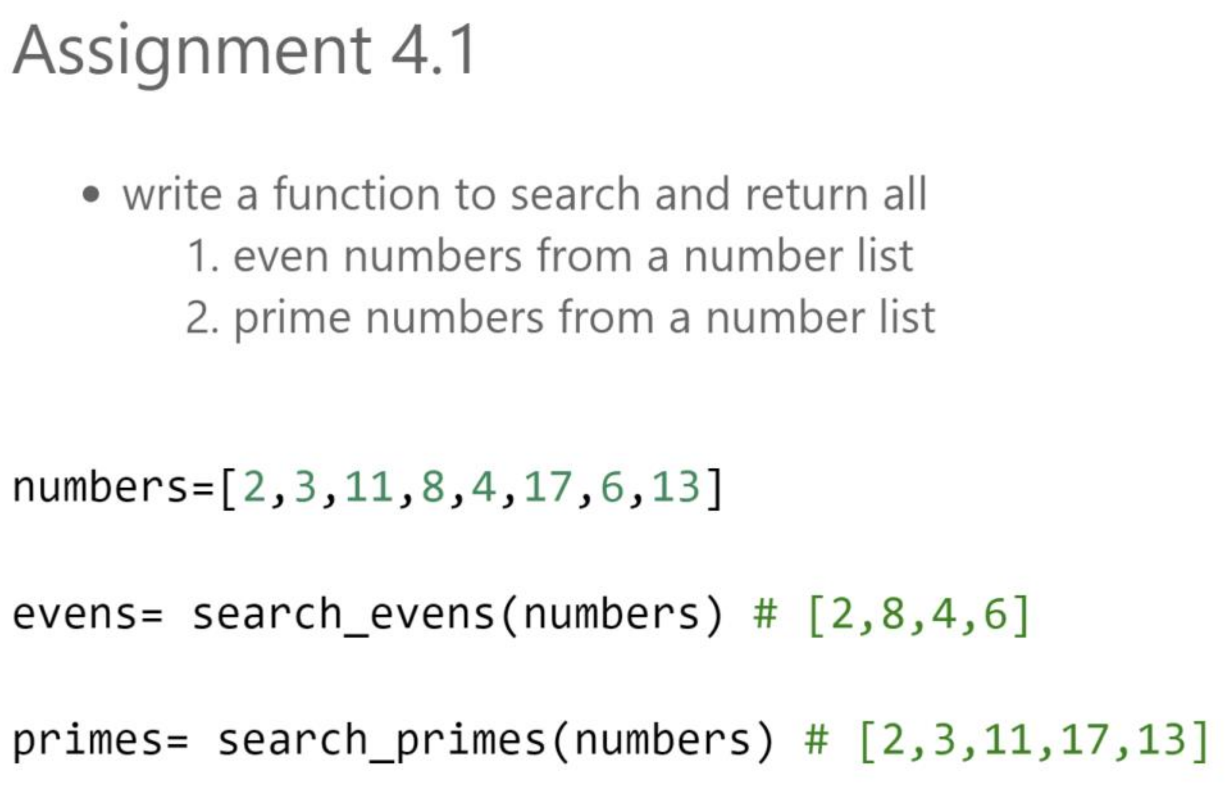
```
def plus(x,y):
    return x+y
```

```
add=plus

plus=20
```



Assignment 4.1

Thursday, November 23, 2023 11:14 AM



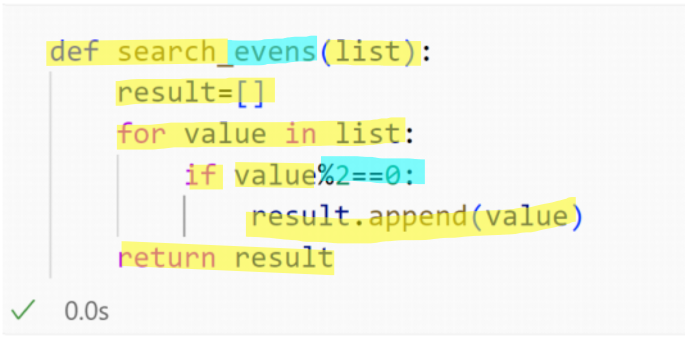
Assignment 4.1

- write a function to search and return all
    1. even numbers from a number list
    2. prime numbers from a number list

```
numbers=[2,3,11,8,4,17,6,13]

evens= search_evens(numbers) # [2,8,4,6]

primes= search_primes(numbers) # [2,3,11,17,13]
```

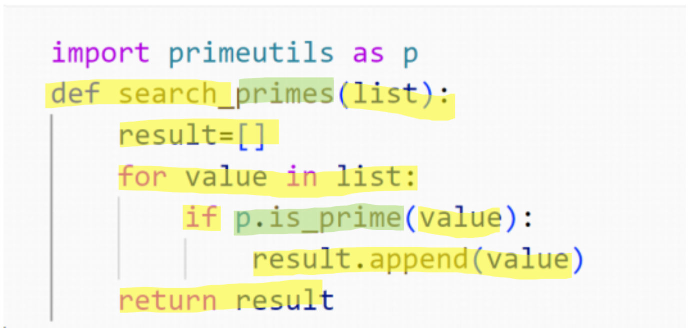
Search

Thursday, November 23, 2023 12:04 PM

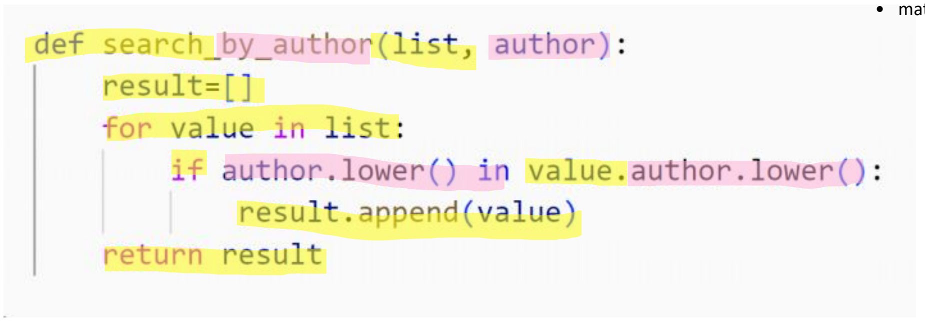


```python
def search_evens(list):
    result=[]
    for value in list:
        if value%2==0:
            result.append(value)
    return result
```

✓ 0.0s



```python
import primeutils as p
def search_primes(list):
    result=[]
    for value in list:
        if p.is_prime(value):
            result.append(value)
    return result
```



```python
def search_by_author(list, author):
    result=[]
    for value in list:
        if author.lower() in value.author.lower():
            result.append(value)
    return result
```

What is the Observation?

- Partially Redundant code.
    - The code is not fully redundant
    - There are small portion which varies in each case
- If it were fully redundant then we needed only one function.
    - since part of the code changes, we need to write it all over again.
- We can't have one search function do all three things.

Why do we have partial Redundancy?

- What is NOT redundant?

What are the steps in the Search?

1. create an empty result list
2. loop through each item
3. Check if ______________
4. if true add to value to list
5. Return the list.

What does Yellow Represent?

- core search logic.
- steps of search
- how to search?

What does Green Represent?

- represents what to search
- matching condition / matching function

How to write a generic Search Function

1. Write only how to search in the search function
    - Don't include Matching Logic Here.

2. Matching Logic can be another function
    - is_prime
    - is_even
    - is_book_by_vivek

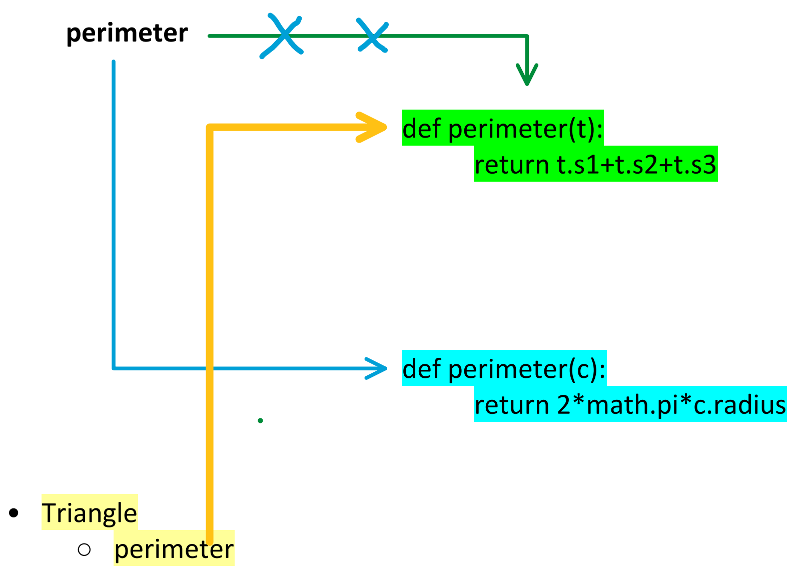
3. In python functions are objects
    - Like any other objects a function can be passed as parameter to another function.

```
def search(  list,match ):
    result=[]
    for value in list:
        if match(value):
            result.append(value)
    return result


primes = search(numbers, p.is_prime)
```

Class as Module

Thursday, November 23, 2023 2:11 PM

```
def perimeter(t):
        return t.s1+t.s2+t.s3


class Triangle:
        pass

Triangle.perimeter=perimeter


def perimeter(c):
        return 2*math.pi*c.radius
```



```
def perimeter(t):
        return t.s1+t.s2+t.s3
```

```
def perimeter(c):
        return 2*math.pi*c.radius
```

- Now the global "perimeter" reference refers to the perimeter(c)
- But Triangle.perimeter refers to perimeter(t)

Assignment 4.6

Thursday, November 23, 2023 5:27 PM

- create a class to define a LinkedList
- It should support basic operations
  - append() to add an item at the end
  - size() to return list size
  - get() to get item from a given index
  - set() to set an item at a given index
  - delete() to delete an item from a given index
  - info() to display all the items in the list

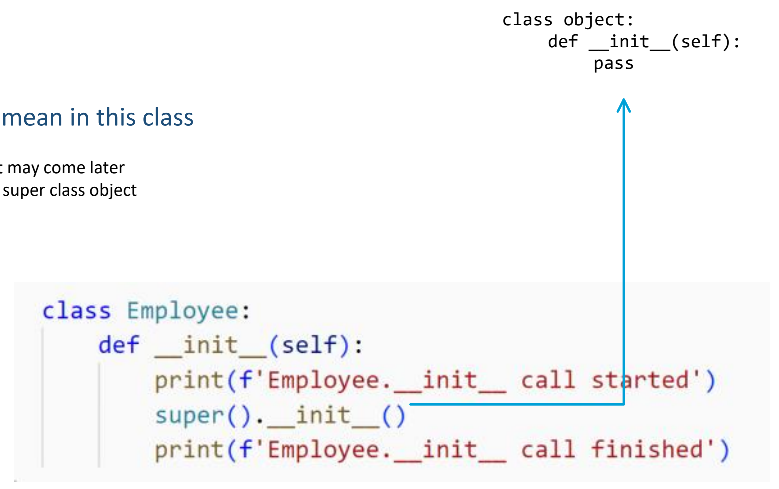
- you are not supposed to use any built-in, third party functions.

MRO

Friday, November 24, 2023 9:53 AM

What does super().__init__() mean in this class

- forget we have a class hierarchy that may come later
- Remember every class has a implicit super class object

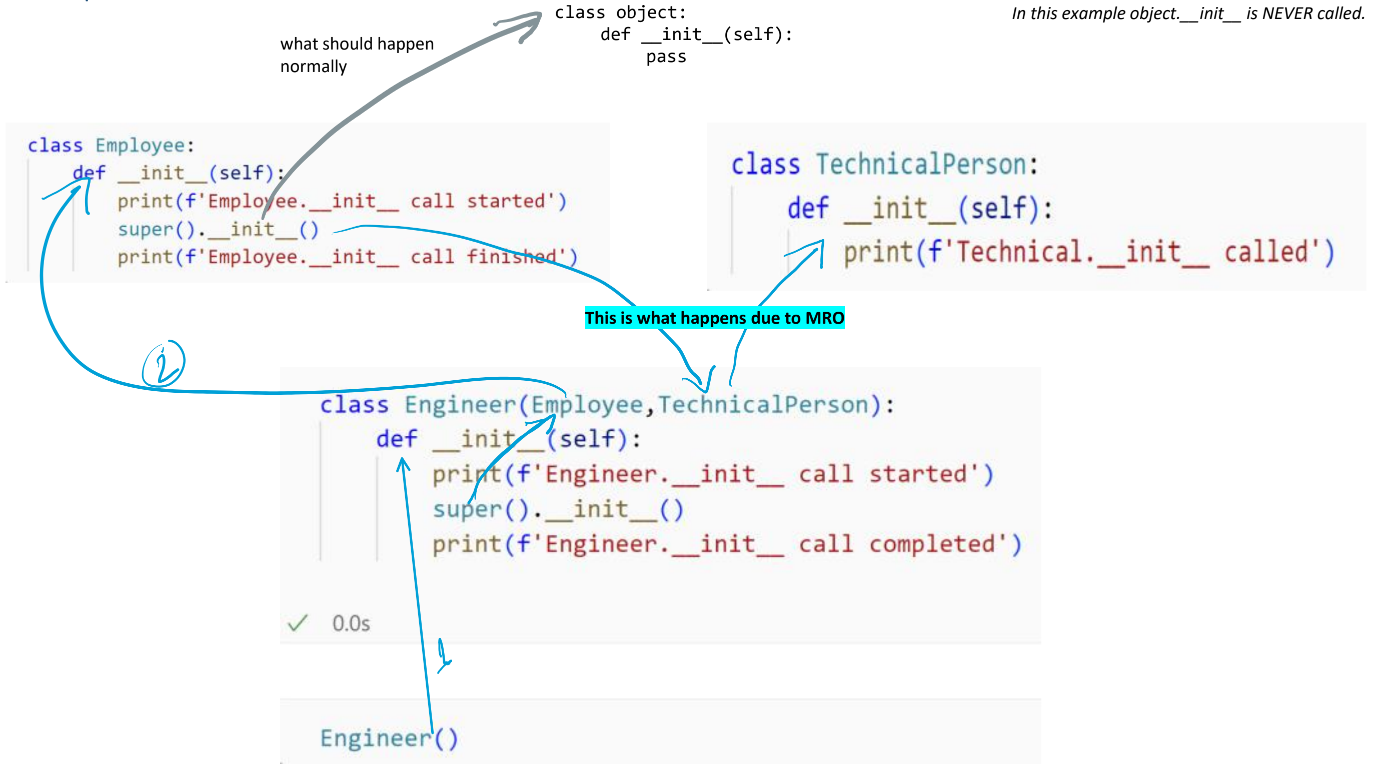


```
class object:
    def __init__(self):
        pass
```

```
class Employee:
    def __init__(self):
        print(f'Employee.__init__ call started')
        super().__init__()
        print(f'Employee.__init__ call finished')
```

In case of Multiple Inheritance

In this example object.__init__ is NEVER called.



```
class object:
    def __init__(self):
        pass
```

```
class Employee:
    def __init__(self):
        print(f'Employee.__init__ call started')
        super().__init__()
        print(f'Employee.__init__ call finished')
```

```
class TechnicalPerson:
    def __init__(self):
        print(f'Technical.__init__ called')
```

```
class Engineer(Employee,TechnicalPerson):
    def __init__(self):
        print(f'Engineer.__init__ call started')
        super().__init__()
        print(f'Engineer.__init__ call completed')
```

✓ 0.0s

```
Engineer()
```

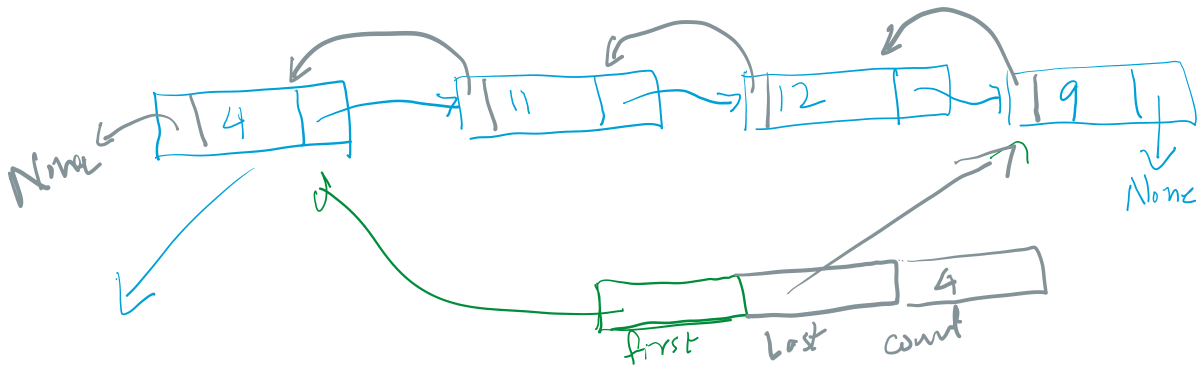
Linked List

Friday, November 24, 2023 11:24 AM

- A linked list is a List of value
- It is logically linear

4	11	12	9

- But for better managment it is stored in scattered memory that are allocated dynamically
- Each value is stored in a different object called Node
- Each node refers to the next one
- Optionally nodes may also refer to previous one.



This is a a Node (Not a Linked List)

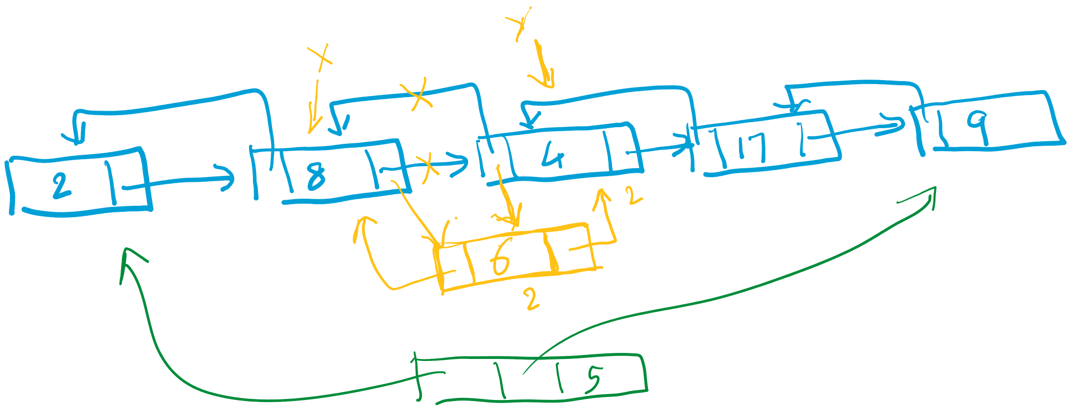
- It represents one unit storage.
- A list will have multiple nodes
- It can refer to the first node to access the entire list

```
class Node:
     def __init__(self, value, next, previous):
          self.value=value
          self.next=next
          self.previous=previous
```

This is the Linked List

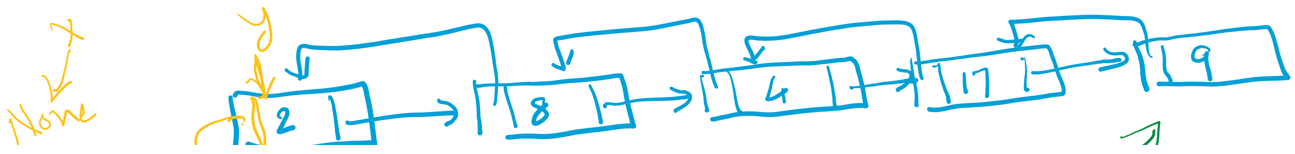
```
class LinkedList:
     def __init__(self):
          self.first=None
          self.last=None
          self.count=0

     def append(self, value)
          pass

     ...
```

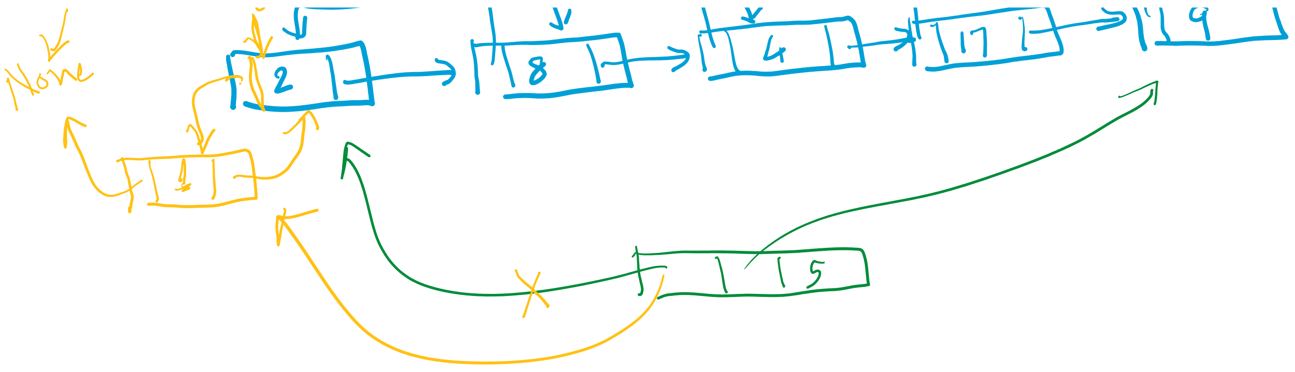
Inserting a node between other Nodes.



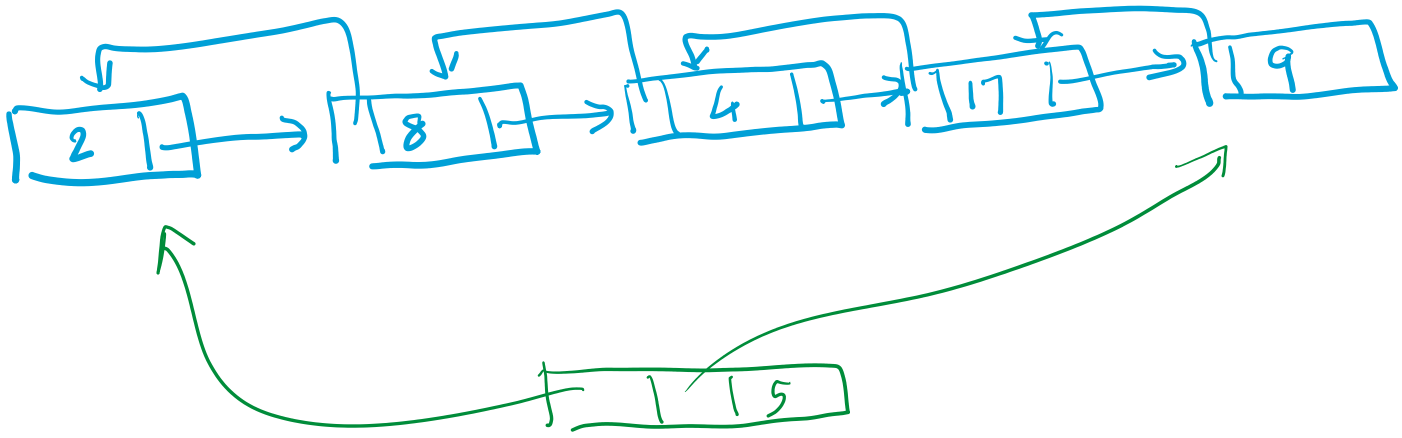
```
new_node= Node(value, previous=x, next=y)
x.next= new_node
y.previous=new_node
```

Insert At begining





```
new_node=Node(value, previous=None, next=y)
y.previous=new_node
self.__first=new_node
```



2
8
4
17
9
5