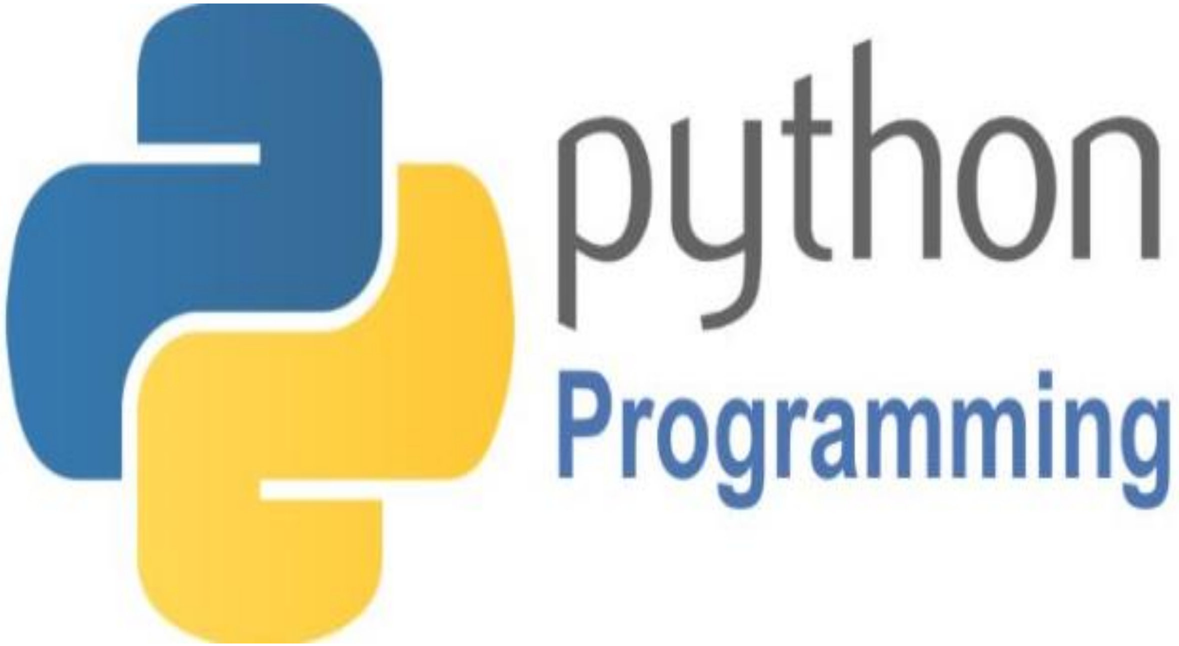
Covenant University
Raising a new Generation of Leaders

PET328

COMPUTER APPLICATIONS IN PETROLEUM ENGINEERING

PET328: COMPUTER APPLICATIONS IN PETROLEUM ENGINEERING
(With Python Programming)



Olatunde O. Mosobalaje (PhD)

Department of Petroleum Engineering,
Covenant University, Ota
Nigeria

OUTLINE

The Appetizer – a presentation

ACQUIRING NASCENT SKILLS FOR EMERGING OIL AND GAS OPPORTUNITIES: DATA ANALYTICS, MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

The Toolbox

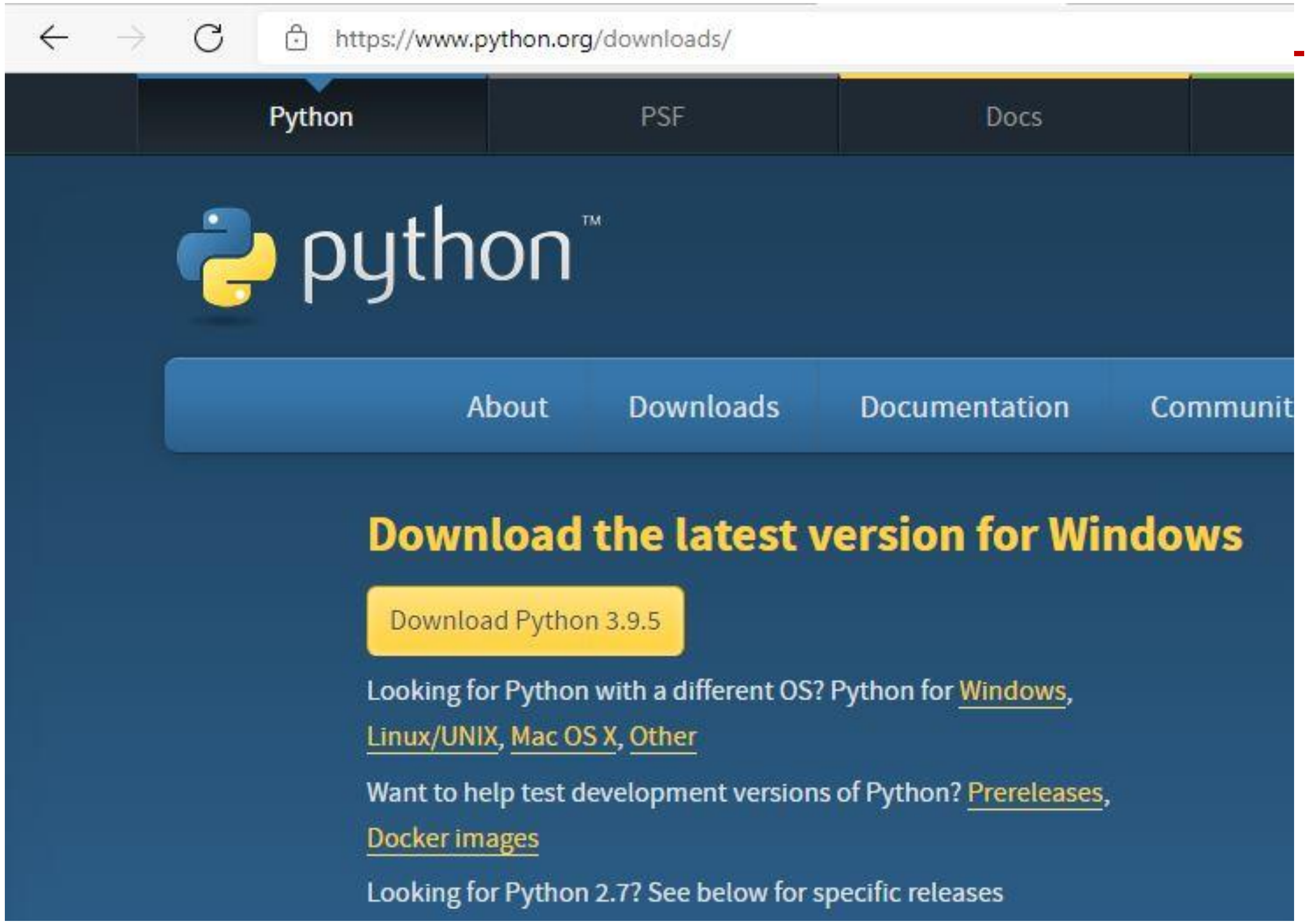
- For this course, the following tools would be needed:
  - Python 3
  - Python Integrated Development and Learning Environment (IDLE)
  - Git and GitHub

PREAMBLES

The Toolbox

- Installing Python 3

To install the latest release of Python 3, go to Python download website: https://www.python.org/downloads/



Download the latest version for Windows

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

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

Looking for Python 2.7? See below for specific releases

The Toolbox

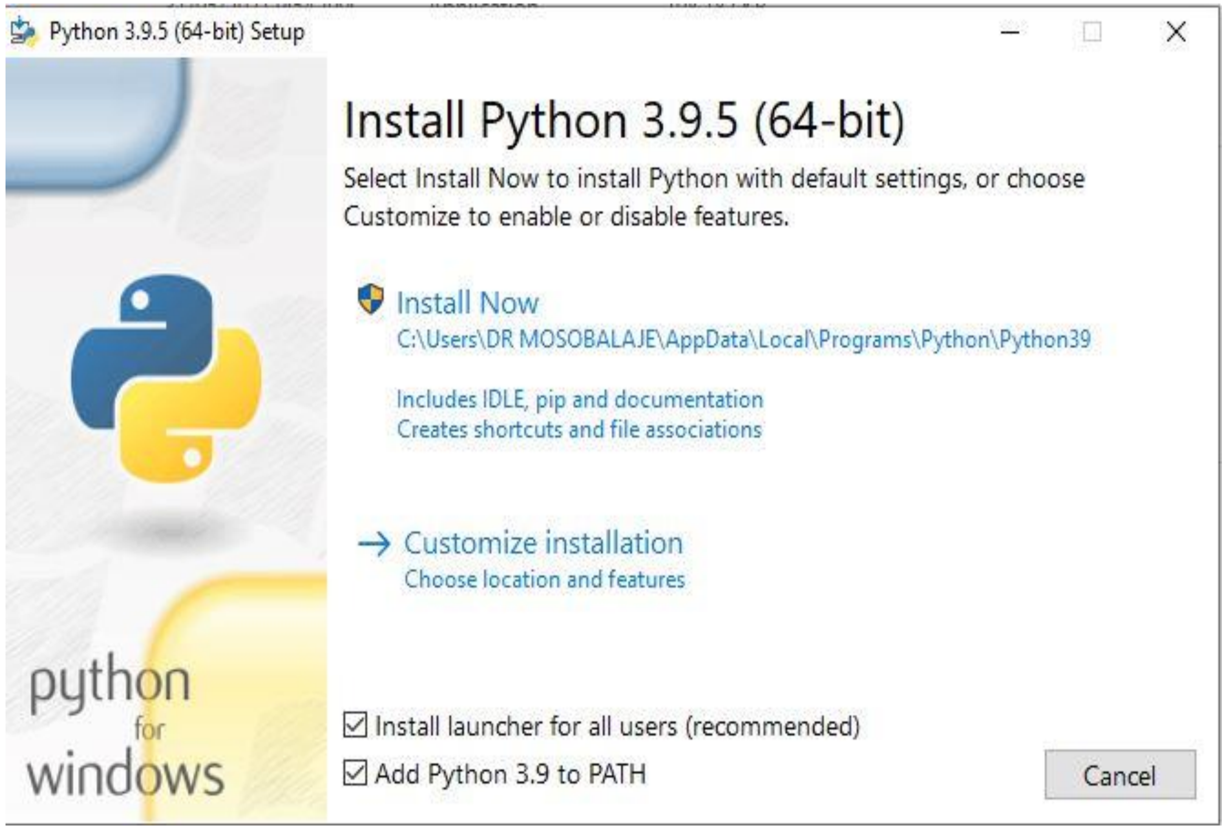
Installing Python 3

Launch the downloaded executable file by double-clicking the file in your download folder.

Follow the steps as the installer leads

Click on the default installation option.

Ensure to check the Add Python 3.9 to PATH


Python 3.9.5 (64-bit) Setup

Install Python 3.9.5 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\DR MOSOBALAJE\AppData\Local\Programs\Python\Python39

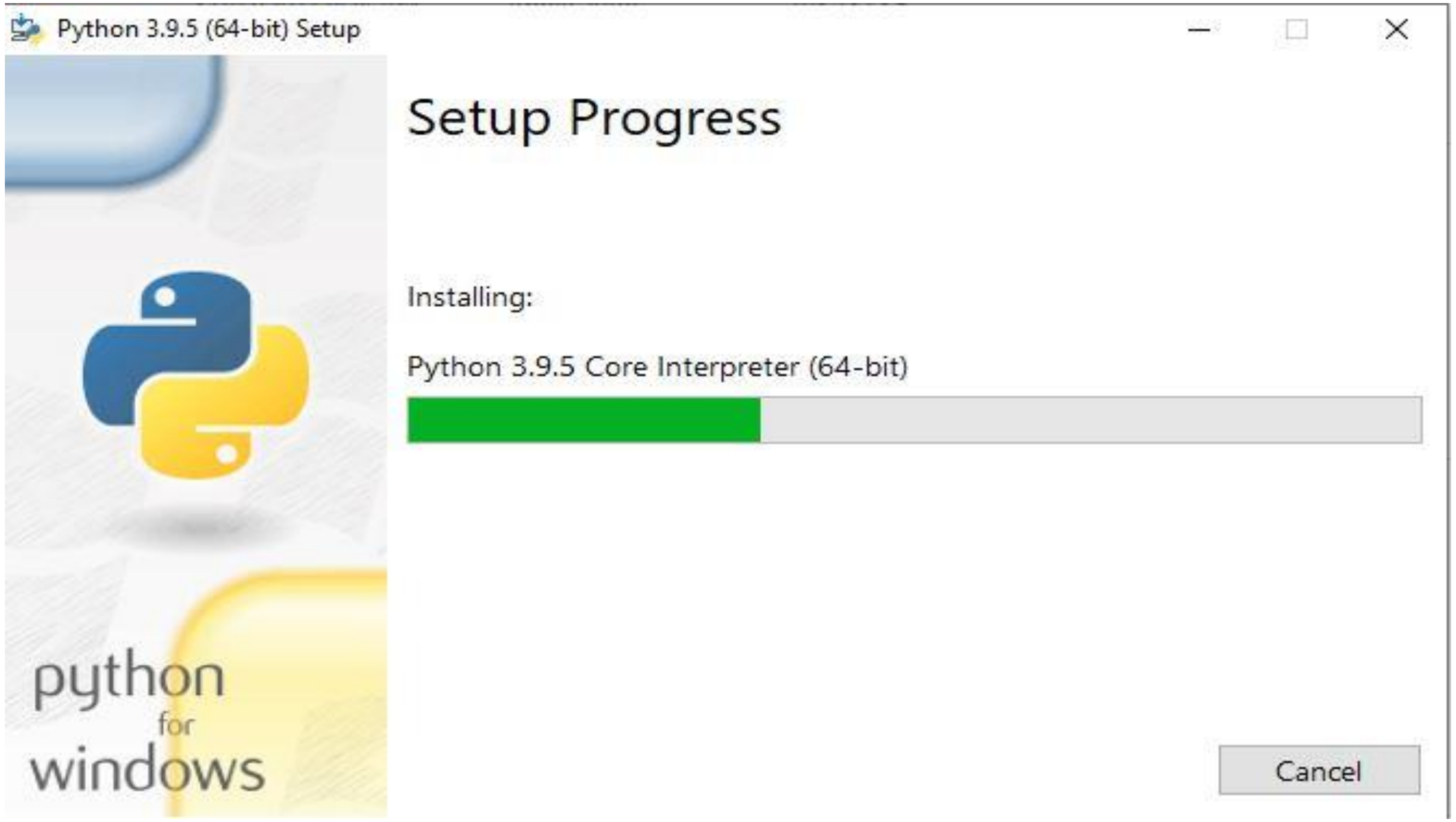
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.9 to PATH

The Toolbox

- Installing Python 3


Python 3.9.5 (64-bit) Setup

Setup Progress

Installing:

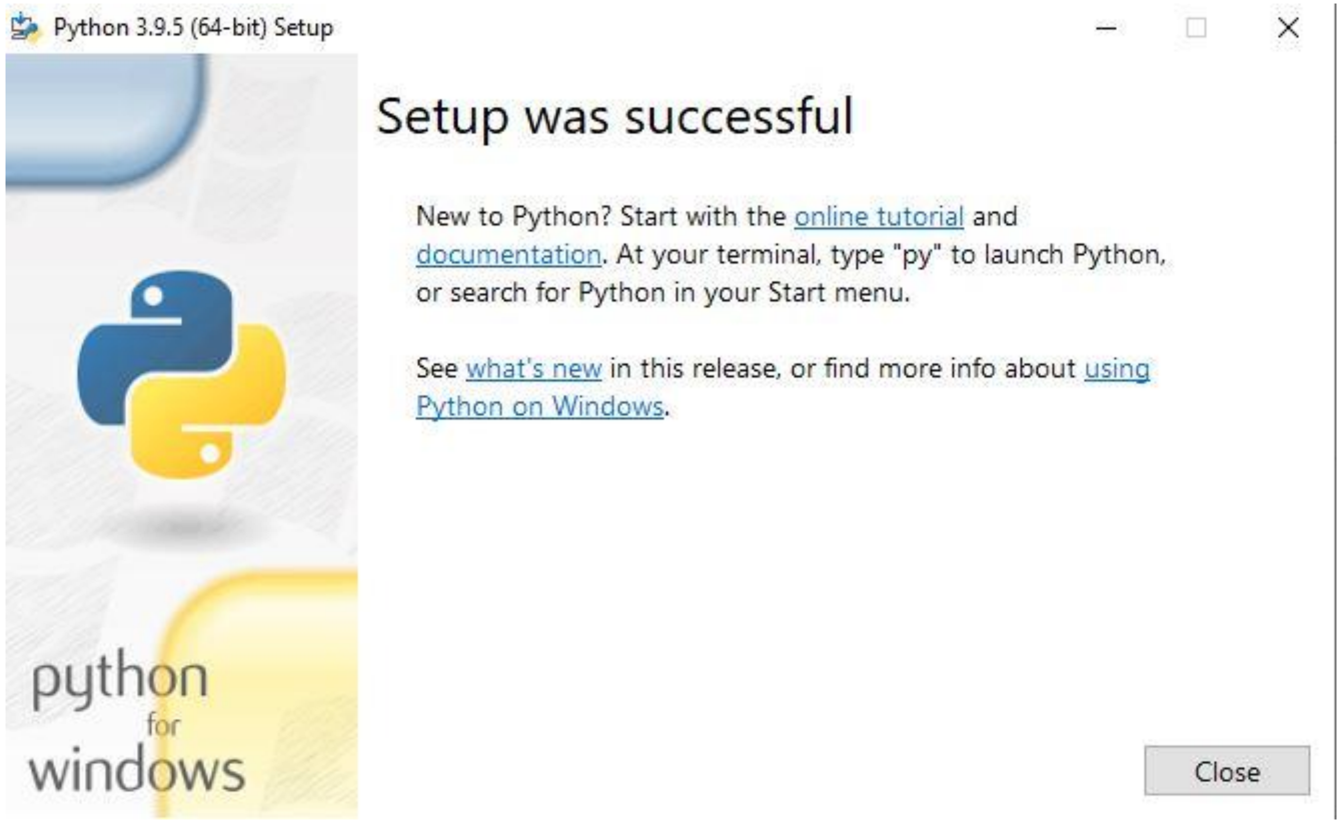
Python 3.9.5 Core Interpreter (64-bit)

PREAMBLES

The Toolbox

Installing Python 3

Click the close button when the installation is completed



Setup was successful

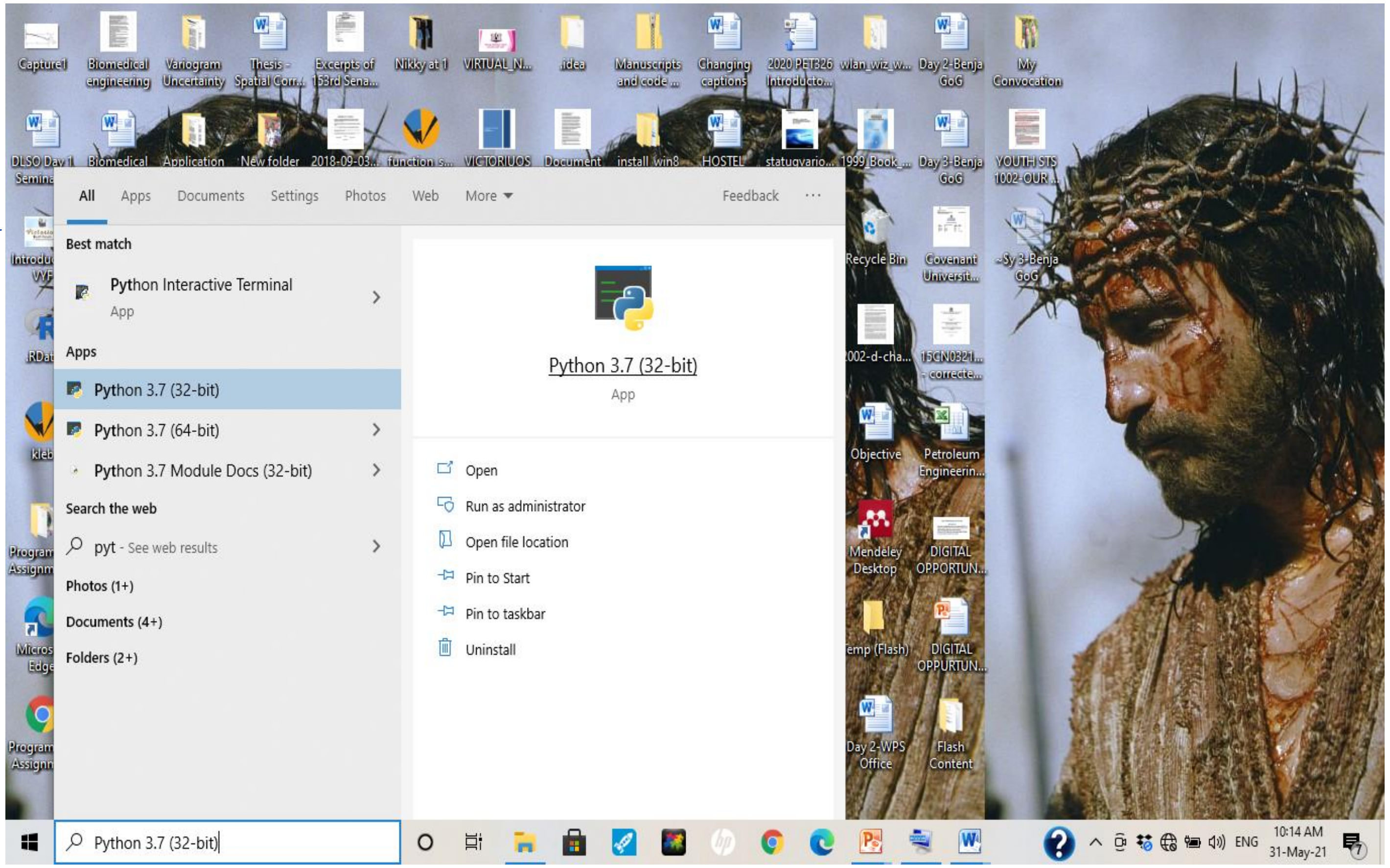
New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See what's new in this release, or find more info about using Python on Windows.

The Toolbox

Launching Python 3

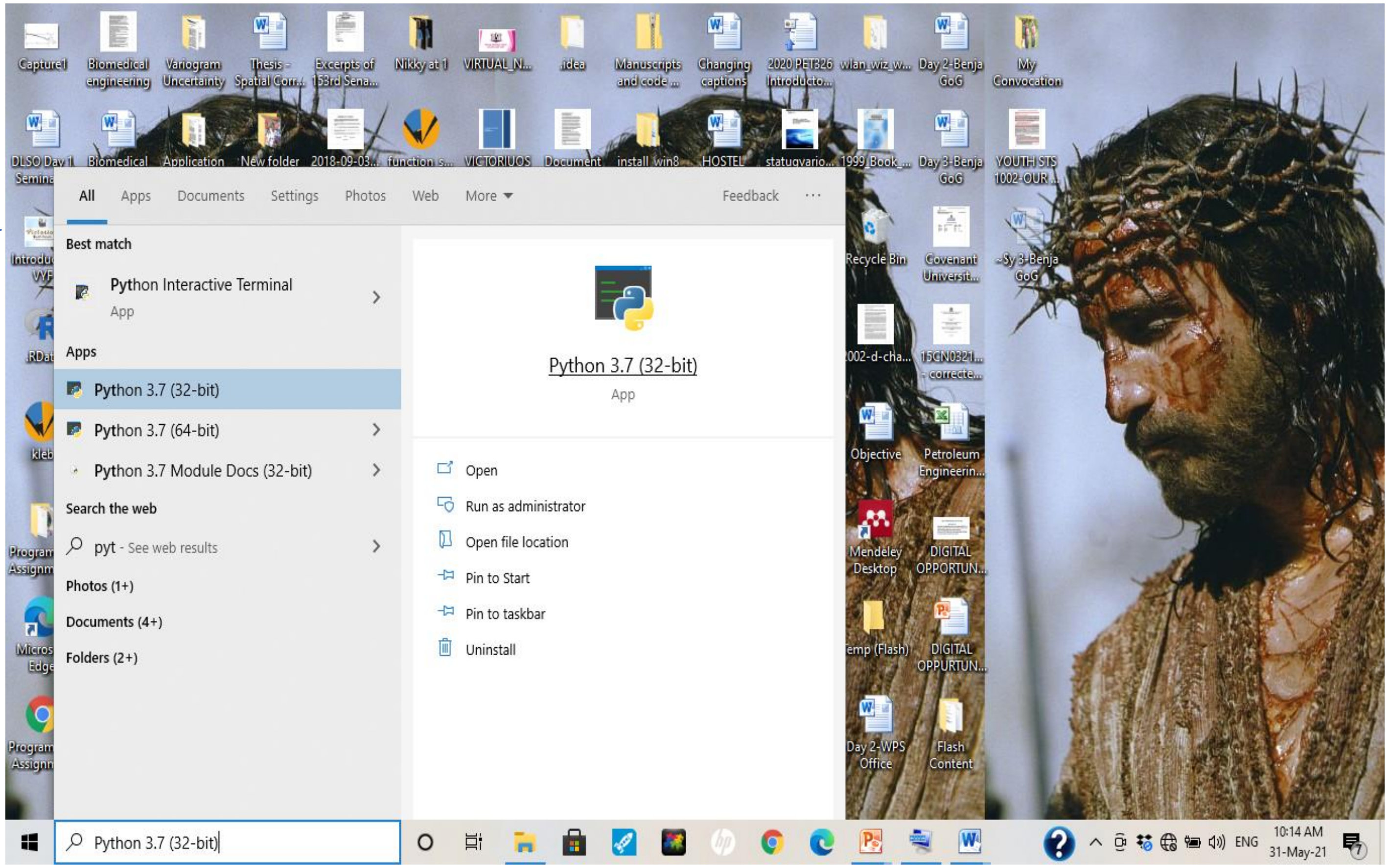
Simply type Python into the Start Menu search box and click the Python program.



The Toolbox

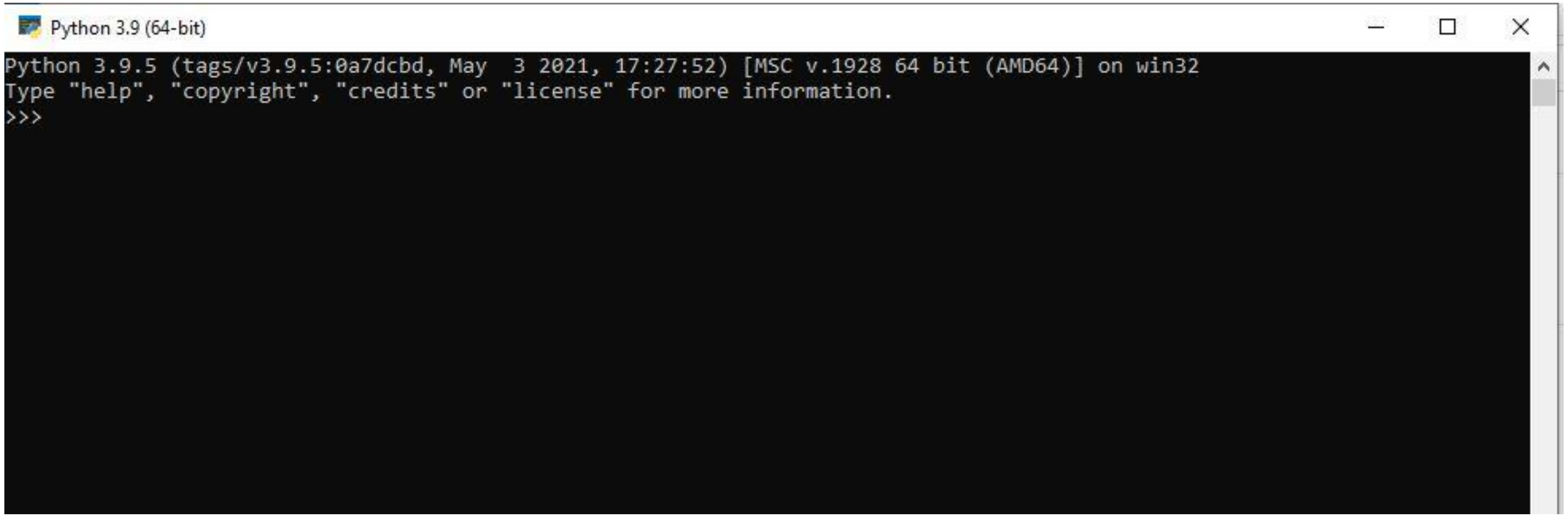
Launching Python 3

Simply type Python into the Start Menu search box and click the Python program.



The Toolbox

Launching Python 3



Python 3.9 (64-bit)

```
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The Toolbox

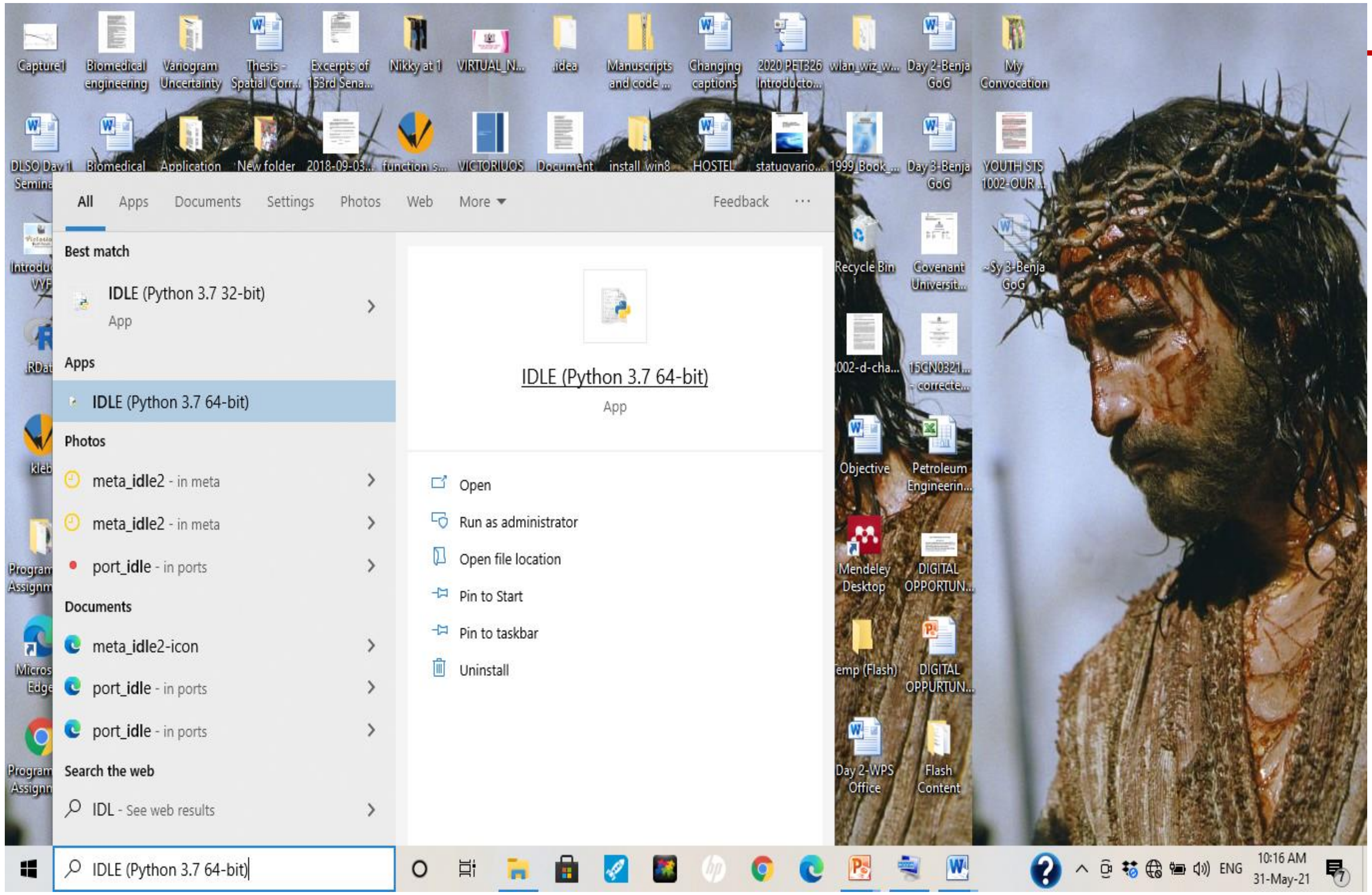
- Python IDLE

Now, the Python DOS-like environment seems boring. Good enough, we will typically not be working on that platform; rather we will interact with Python from a platform known as Interactive Development and Learning Environment (IDLE)

The Toolbox

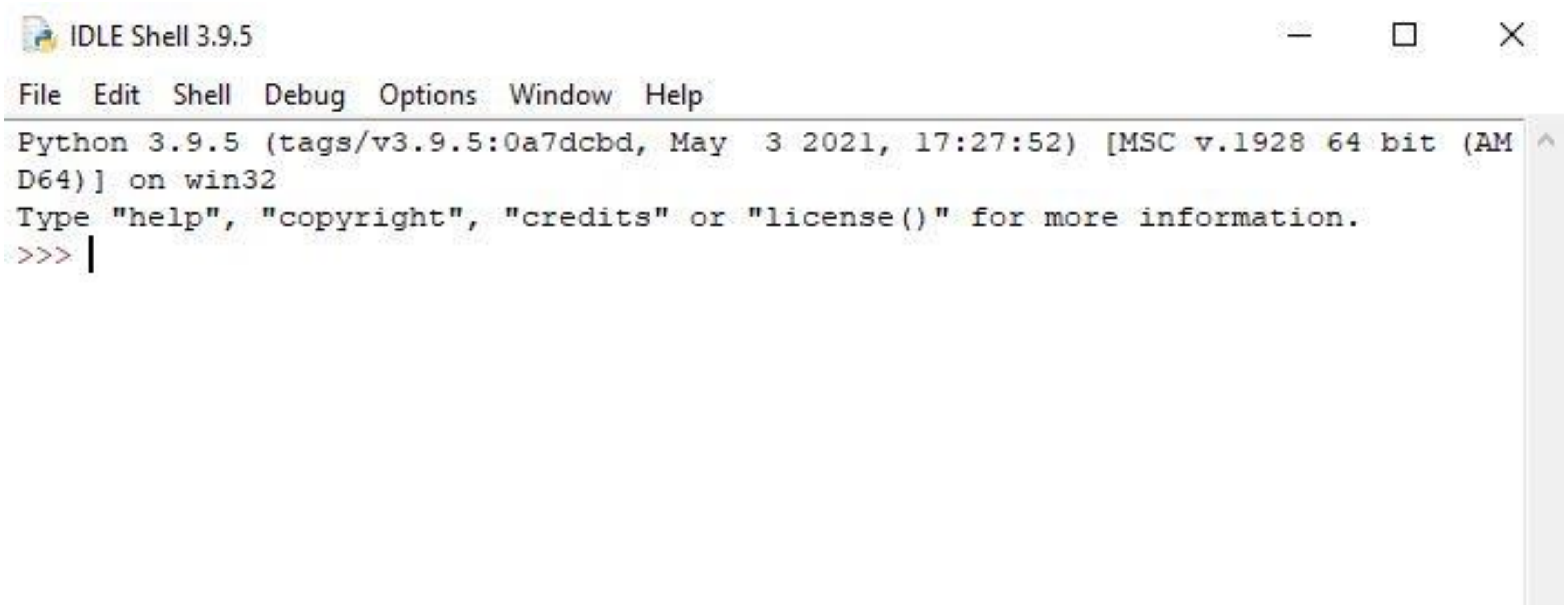
- Python IDLE

To launch IDLE, simply type IDLE into the Start Menu search box and click on the IDLE program.



The Toolbox

Python IDLE



```
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
```

The Toolbox

- Python IDLE

There are two ways by which you could communicate with Python from the IDLE environment:

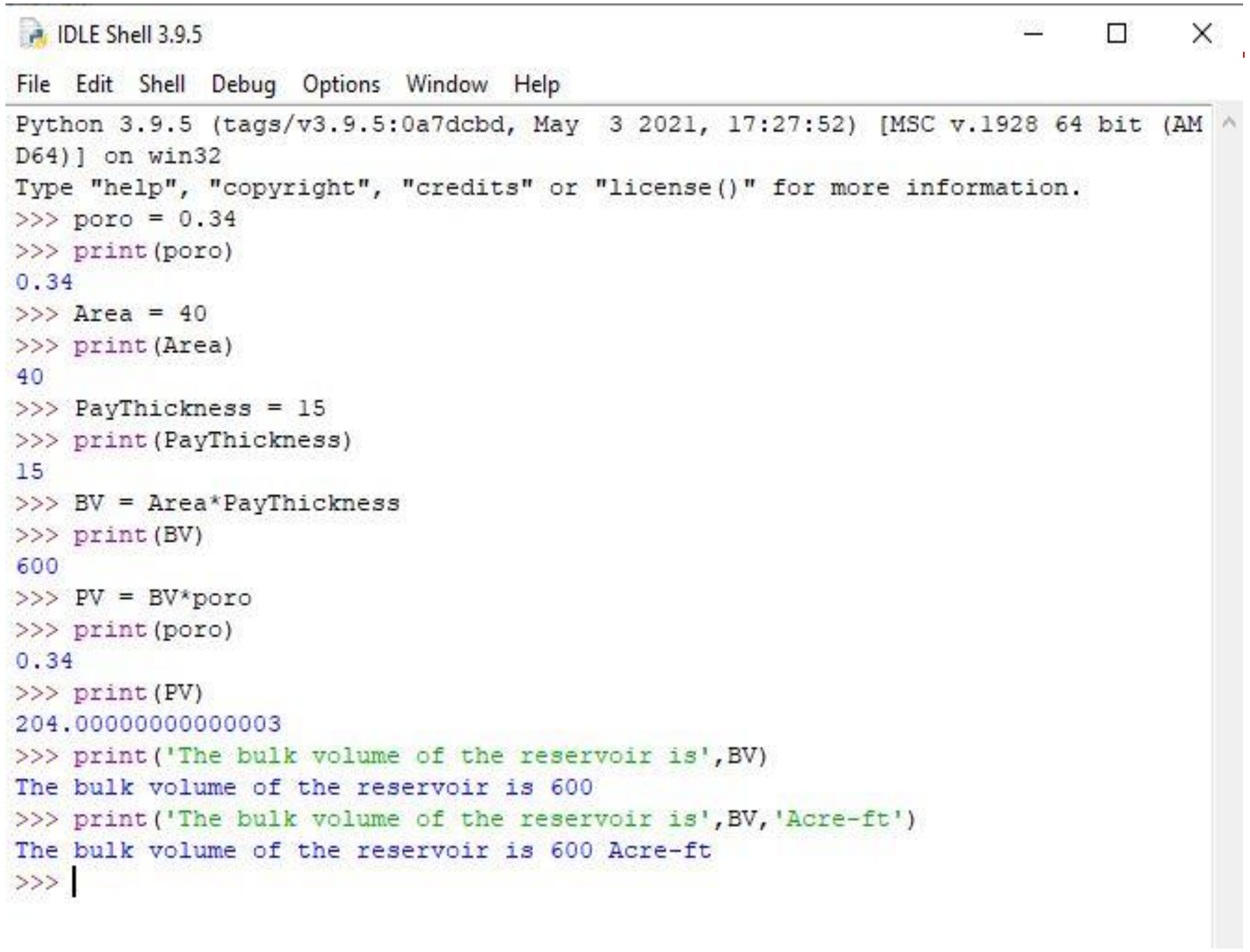
- Interactive
- From a file (script)

PREAMBLES

The Toolbox

- Communicating with Python interactively

In this case, you type in Python command (one at a time) into the console. Each command get executed once the 'Enter' key is pressed. Depending on the command, results may be displayed on the console once the command is executed.

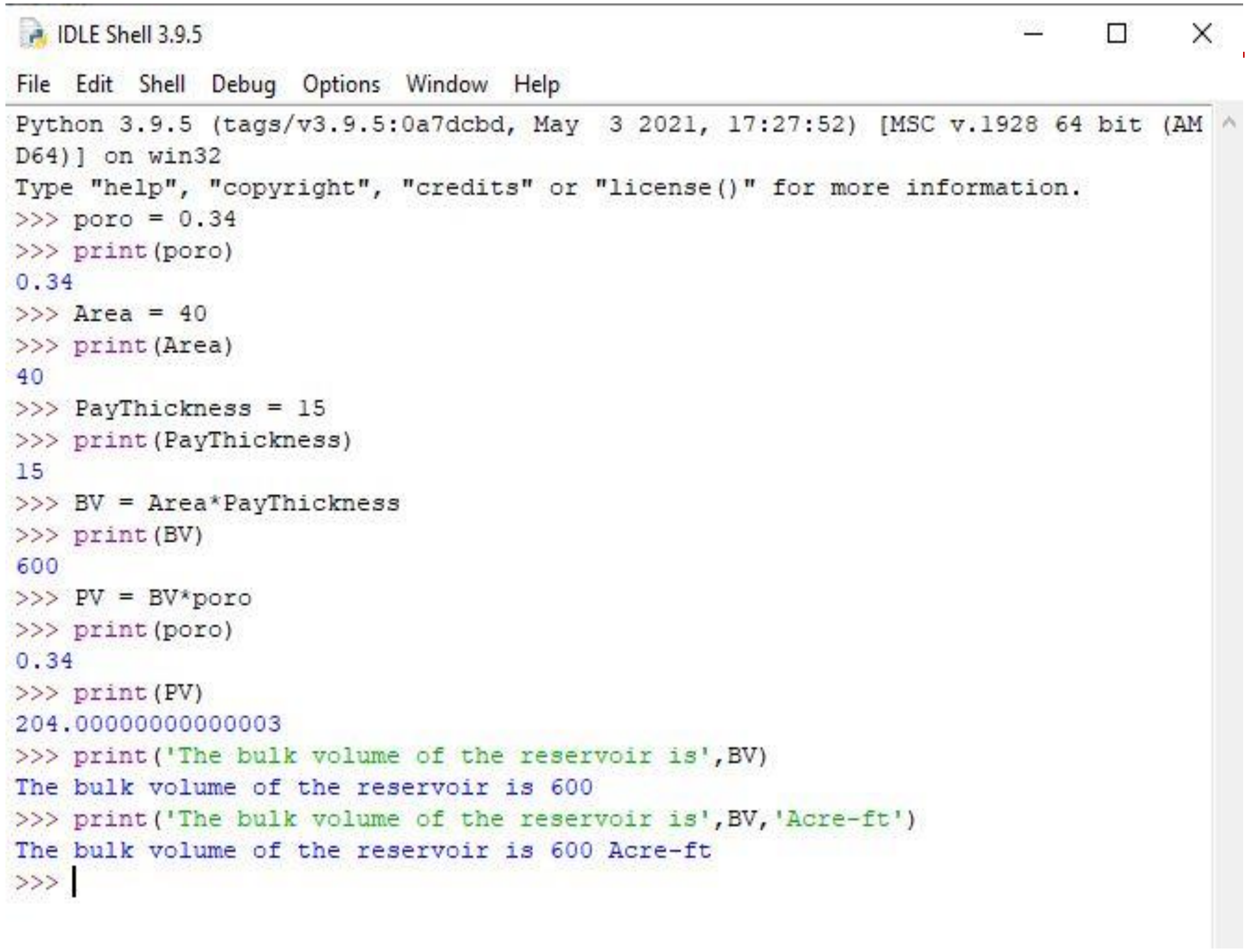


```
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> poro = 0.34
>>> print(poro)
0.34
>>> Area = 40
>>> print(Area)
40
>>> PayThickness = 15
>>> print(PayThickness)
15
>>> BV = Area*PayThickness
>>> print(BV)
600
>>> PV = BV*poro
>>> print(poro)
0.34
>>> print(PV)
204.00000000000003
>>> print('The bulk volume of the reservoir is',BV)
The bulk volume of the reservoir is 600
>>> print('The bulk volume of the reservoir is',BV,'Acre-ft')
The bulk volume of the reservoir is 600 Acre-ft
>>> |
```

The Toolbox

- Communicating with Python interactively

In this case, you type in Python command (one at a time) into the console. Each command get executed once the 'Enter' key is pressed. Depending on the command, results may be displayed on the console once the command is executed.



```
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> poro = 0.34
>>> print(poro)
0.34
>>> Area = 40
>>> print(Area)
40
>>> PayThickness = 15
>>> print(PayThickness)
15
>>> BV = Area*PayThickness
>>> print(BV)
600
>>> PV = BV*poro
>>> print(poro)
0.34
>>> print(PV)
204.00000000000003
>>> print('The bulk volume of the reservoir is',BV)
The bulk volume of the reservoir is 600
>>> print('The bulk volume of the reservoir is',BV,'Acre-ft')
The bulk volume of the reservoir is 600 Acre-ft
>>> 
```

The Toolbox

- Communicating with Python from a file

In this case, you type in Python commands (all at a time) into a text file editor (code editor). The commands don't get executed as they are being typed. Rather, they get executed (sequentially) when submitted as a whole to the Python interpreter.

The Toolbox

Communicating with Python from a file

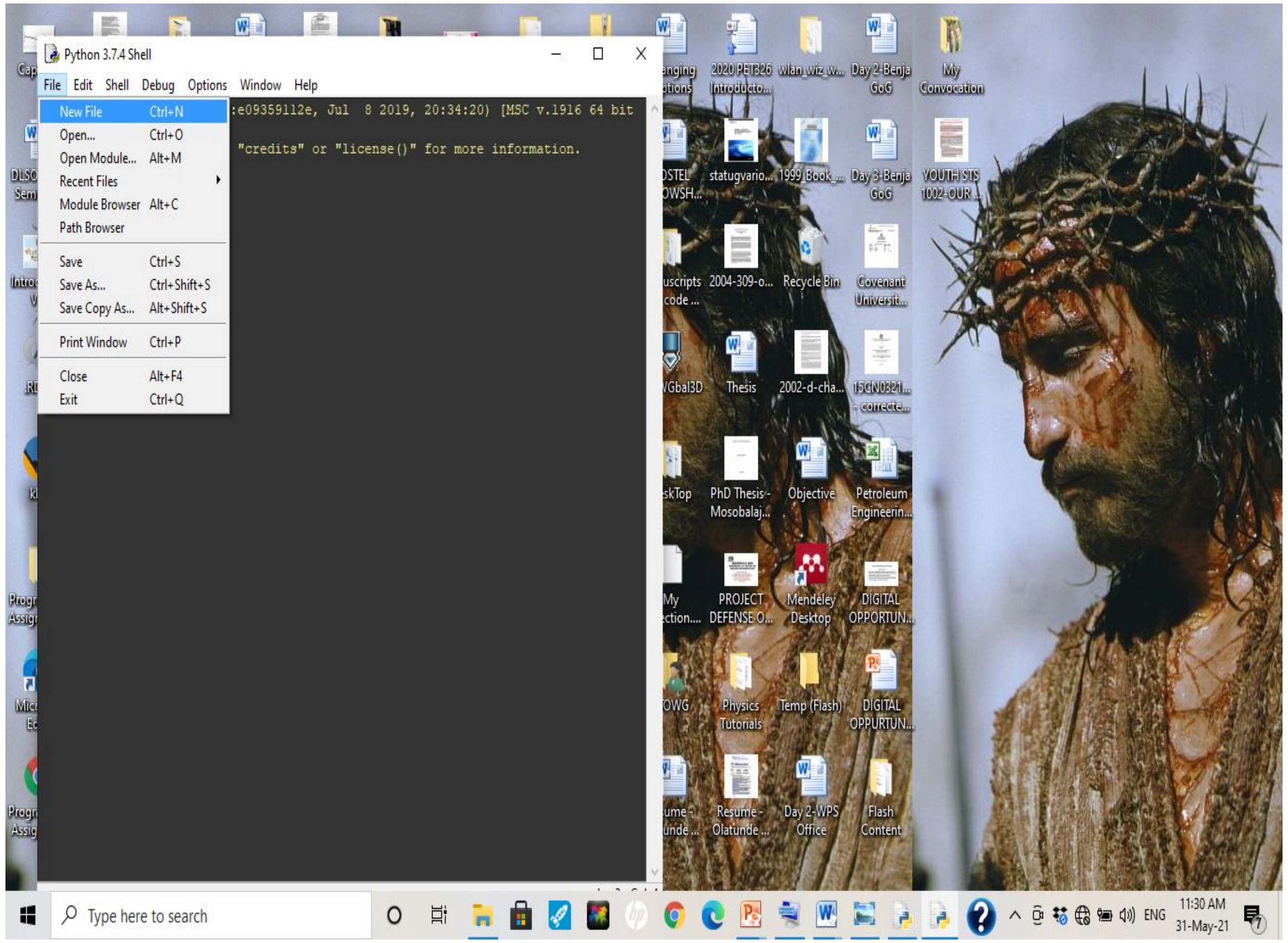
Any text editor program could be used for this purpose, as long as the file is saved as a .py file.

Good, Python has an in-built text editor for this purpose.

The Toolbox

- Communicating with Python from a file

To launch Python's in-built code editor, just click on the File menu and choose New File.

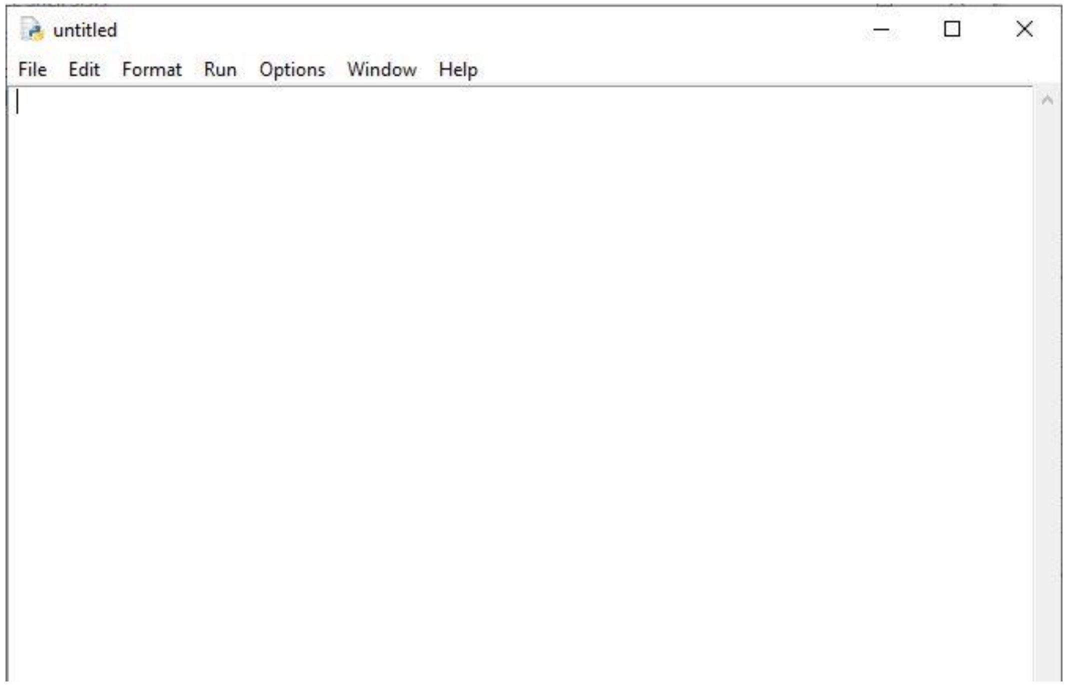


```
:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit

"credits" or "license()" for more information.
```

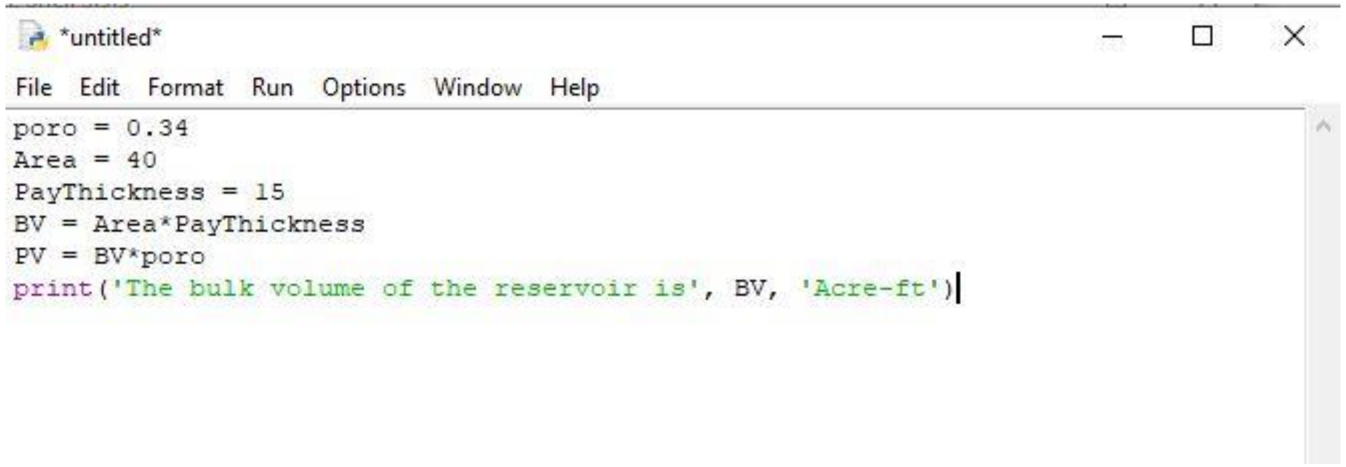
PREAMBLES

The Toolbox

- Communicating with Python from a file



Once the editor is opened, you can type in your lines of codes.



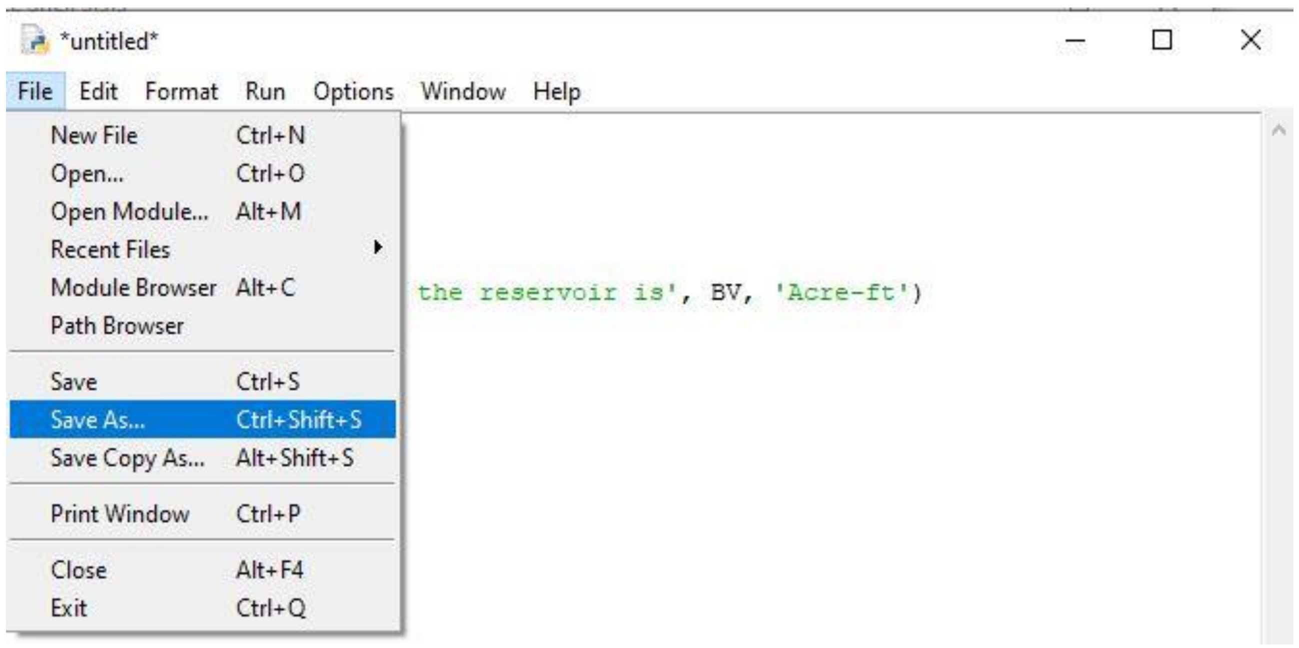
```
poro = 0.34
Area = 40
PayThickness = 15
BV = Area*PayThickness
PV = BV*poro
print('The bulk volume of the reservoir is', BV, 'Acre-ft')
```

The Toolbox

Communicating with Python from a file

Before submitting the lines of codes in the code editor to the Python interpreter, you need to save the editor file.

To save, simply go the File menu and choose Save As



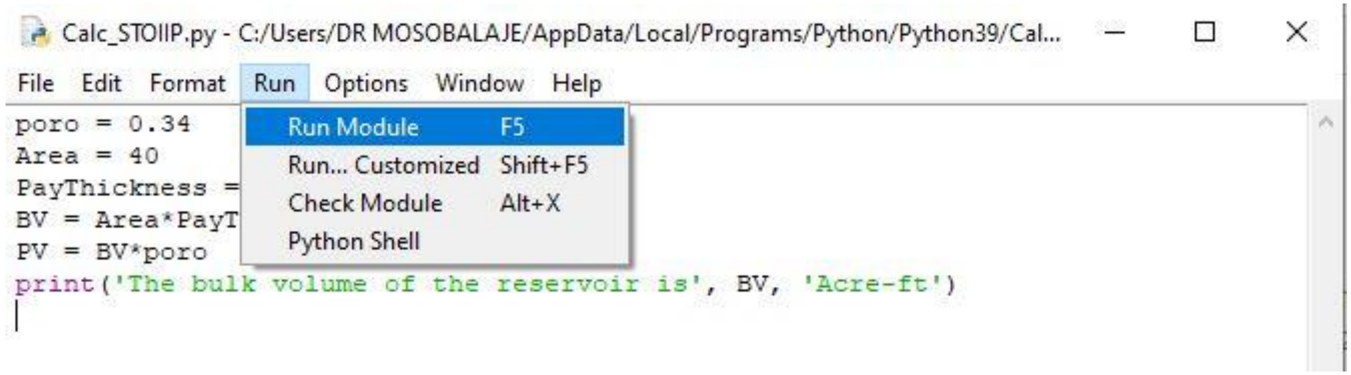
```
the reservoir is', BV, 'Acre-ft')
```

The Toolbox

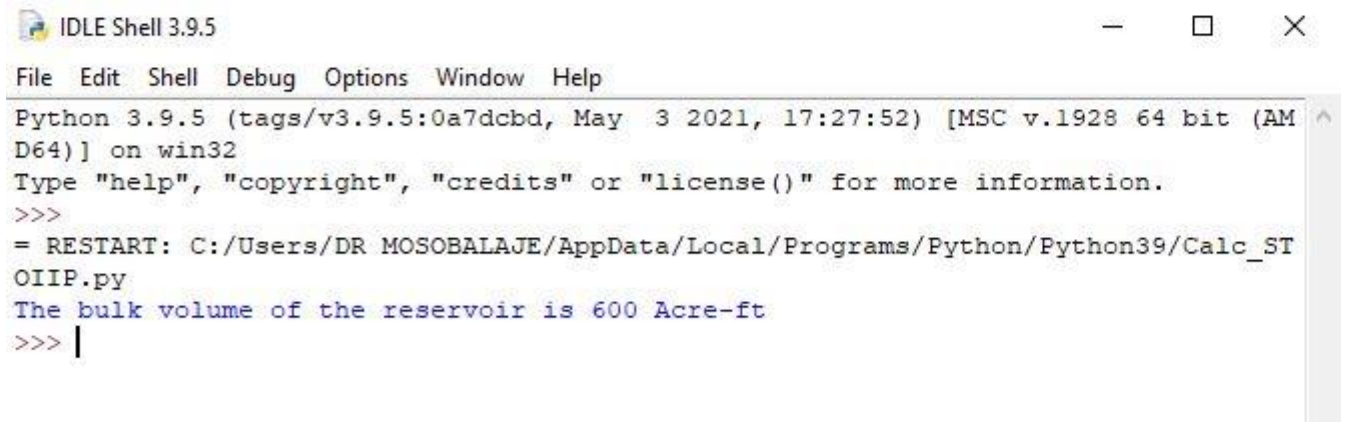
Communicating with Python from a file

Once the file (script) is saved, the code lines can be submitted to the Python interpreter by choosing item 'Run Module' in the Run menu.

The output of the code execution (if any) is subsequently displayed on the Python console.


Calc_STOIIP.py - C:/Users/DR MOSOBALAJE/AppData/Local/Programs/Python/Python39/Cal...
File Edit Format Run Options Window Help
Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell

```
poro = 0.34
Area = 40
PayThickness =
BV = Area*PayT
PV = BV*poro
print('The bulk volume of the reservoir is', BV, 'Acre-ft')
```

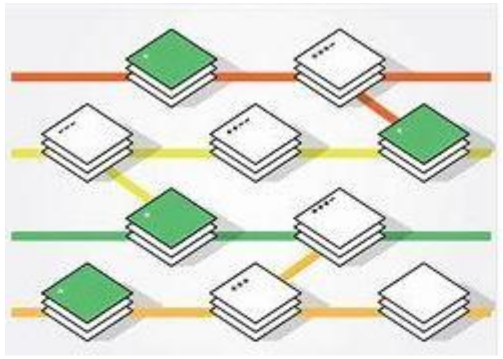


```
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/DR MOSOBALAJE/AppData/Local/Programs/Python/Python39/Calc_ST
OIIP.py
The bulk volume of the reservoir is 600 Acre-ft
>>>
```

The Toolbox

- Git and GitHub

Git is an open source version control software.



What is Version Control?

Version control (VC) is a system used for keeping track of changes made to a file over time. As the changes are made, the system records and save the state of the file at instances indicated by the user. Such user can revert back to a previous version of the file when necessary.

Essentially, the VC system keeps the latest version of the file but also keeps a record of all changes between all versions.

The Toolbox

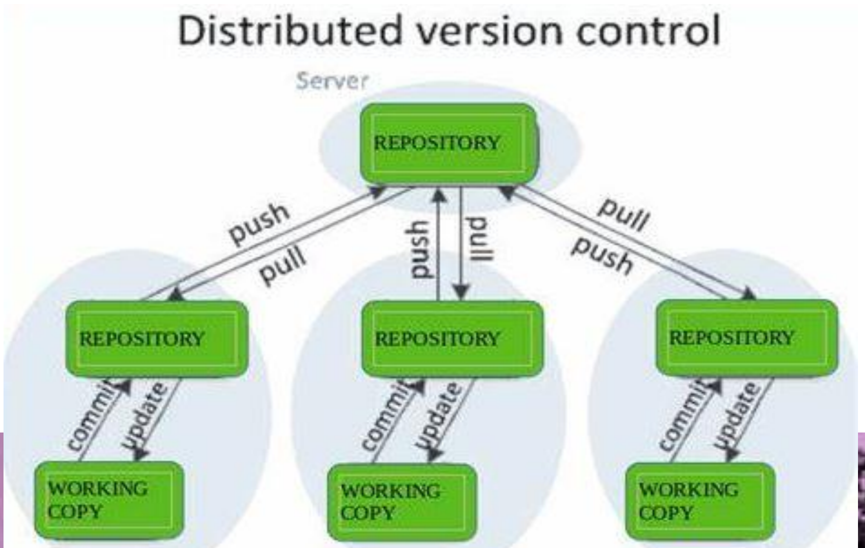
- Git and GitHub

And, there is something called Distributed Version Control (DVC)

What is Distributed Version Control?

Typically, real life projects (including oilfield digital projects) are done by teams whose members need to collaborate – work together on same files. Individual members of the team can make changes to such shared files. There is therefore a need to make such file available on a central server and to keep track of the following:

- who made what change?
- When was the change made?
- Why was the change made?



The Toolbox

Git and GitHub

And, there is something called Distributed Version Control (DVC)

What is Distributed Version Control?

A version control system that also comes with the capabilities for collaboration among several people is known as Distributed Version Control system.

Git is a version control system – locally hosted on your system.

GitHub is an online platform that interfaces with Git, hosting your files on remote servers thereby making them available for collaboration with others.

The Toolbox

Git and GitHub

In this course, we shall be working as a team, therefore, both Git and GitHub are part of tools we shall be using. Essentially, submissions to some assignments shall be in the form of code file editing and sharing between students and the Course Instructor.

Assignment 1

Get the following tools ready on your PC:

- Git - install
- A user account on github.com
- GitHub desktop - install

PREAMBLES

The Embedded Course

A Coursera course is embedded into this course (PET328). It is compulsory that all students completes the Coursera course as it is part of the assessment items in PET328.

coursera
Programming for Everybody (Getting Started with Python)
Categories: Arts and Humanities
Business
Computer Science
Data Science
Information Technology
Healt
Welcome to
Covenant University on Coursera
Start learning on Coursera!

The Embedded Course

The embedded course is titled 'Programming for Everybody (Getting Started with Python).

The course is offered by University of Michigan.

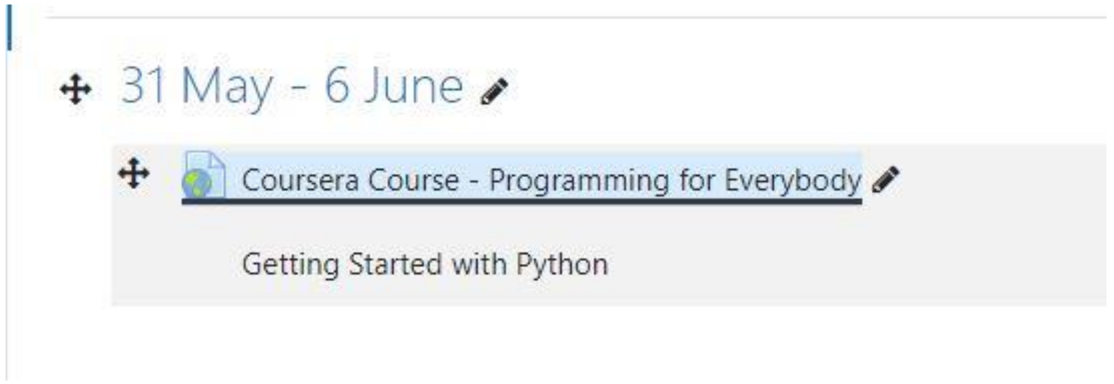
Programming for Everybody (Getting Started with Python)

Offered by

UNIVERSITY OF MICHIGAN

The Embedded Course

The link to the embedded course has been added to the PET328 course site on Moodle. To enroll for the Coursera course, simply click on the link.

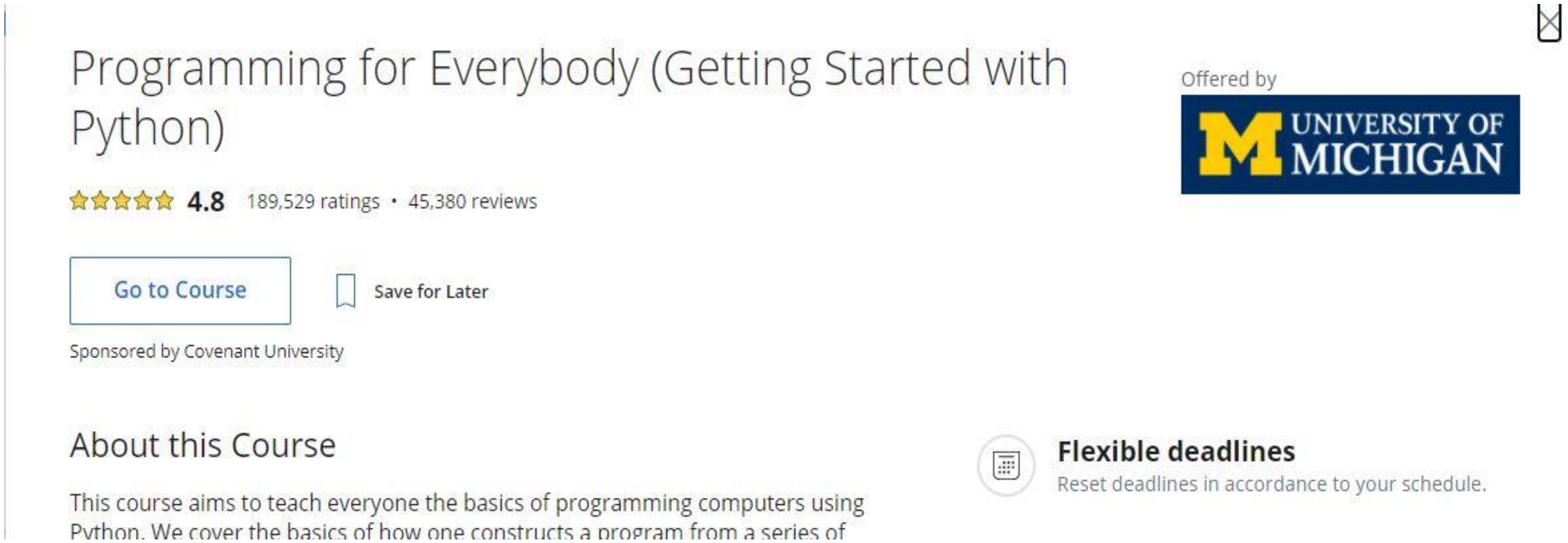

31 May - 6 June

Coursera Course - Programming for Everybody

Getting Started with Python

PREAMBLES

The Embedded Course



Programming for Everybody (Getting Started with Python)

4.8 189,529 ratings • 45,380 reviews

Sponsored by Covenant University

Offered by UNIVERSITY OF MICHIGAN

About this Course

This course aims to teach everyone the basics of programming computers using Python. We cover the basics of how one constructs a program from a series of

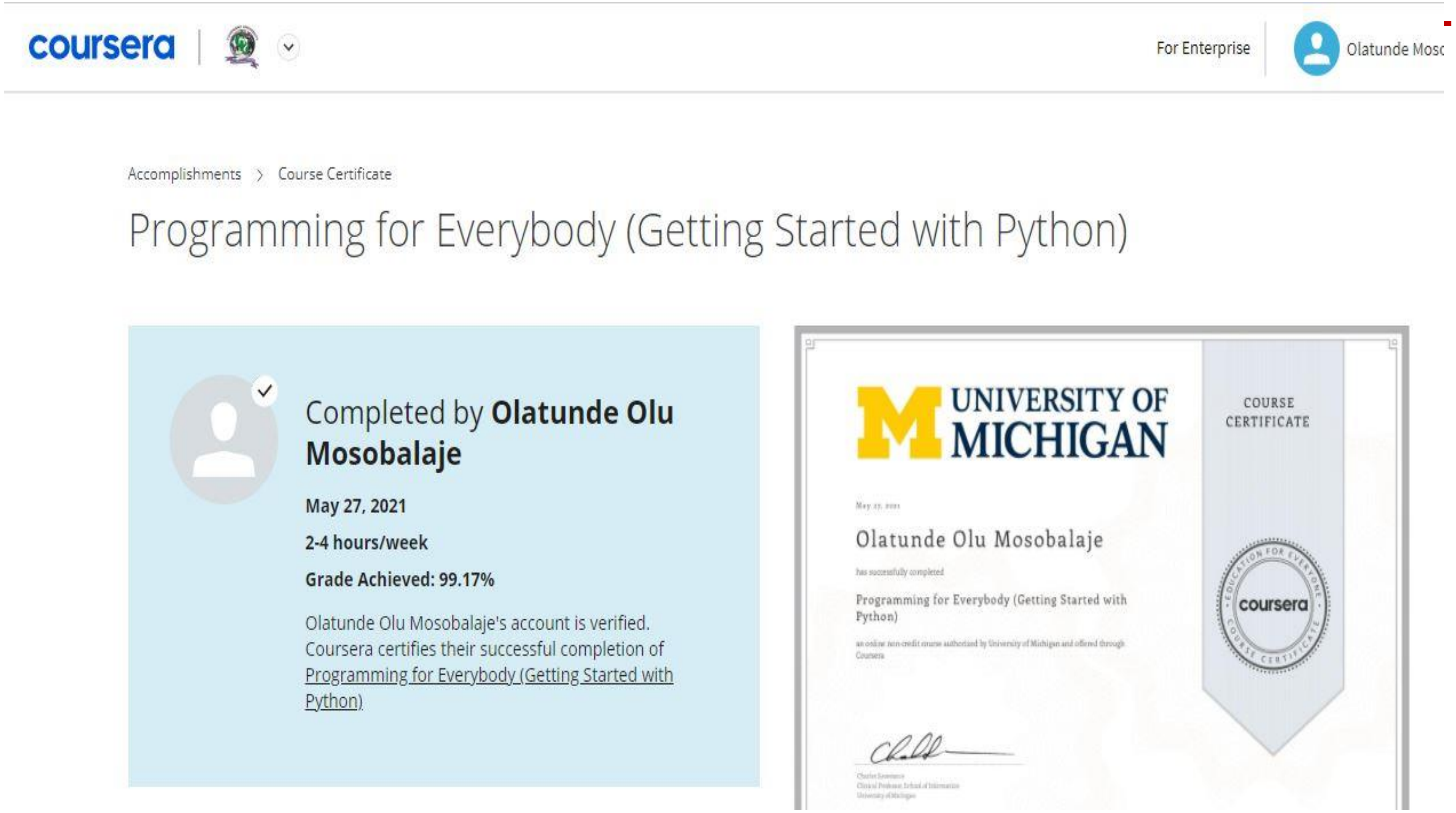
Flexible deadlines

Reset deadlines in accordance to your schedule.

PREAMBLES

The Embedded Course

- When you complete the course, you are awarded a certificate of completetion!!!



Programming for Everybody (Getting Started with Python)

Completed by **Olatunde Olu Mosobalaje**

May 27, 2021

2-4 hours/week

Grade Achieved: 99.17%

Olatunde Olu Mosobalaje's account is verified. Coursera certifies their successful completion of Programming for Everybody (Getting Started with Python)

UNIVERSITY OF MICHIGAN

COURSE CERTIFICATE

May 27, 2021

Olatunde Olu Mosobalaje

has successfully completed

Programming for Everybody (Getting Started with Python)

an online non-credit course authorized by University of Michigan and offered through Coursera

Charles Severance
Clinical Professor, School of Information
University of Michigan

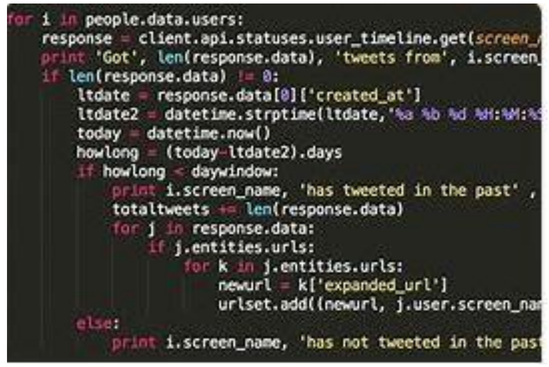
Introduction to Computer Programming

- Analogy: Programming language vs. Natural language
  - There is a striking similarity between learning programming language and learning natural language. In both cases, the process is thus:
    - learn the vocabulary and the grammar – spell words, construct sentences etc.
    - Communicate
      - natural language: use words, sentences, paragraphs to communicate an idea
      - Programming language: use keywords, variables, functions, expressions, statements to communicate steps to computer.

PREAMBLES

Introduction to Computer Programming

- A program is simply a collection of sequential Python statements written to perform a specific task



```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_
    print 'Got', len(response.data), 'tweets from', i.screen_
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
        ltdate2 = datetime.strptime(ltdate,'%a %b %d %H:%M:%S
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past' ,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
                    for k in j.entities.urls:
                        newurl = k['expanded_url']
                        urlset.add((newurl, j.user.screen_nam
        else:
            print i.screen_name, 'has not tweeted in the past
```

Introduction to Computer Programming

Fundamental patterns (concepts) in a program

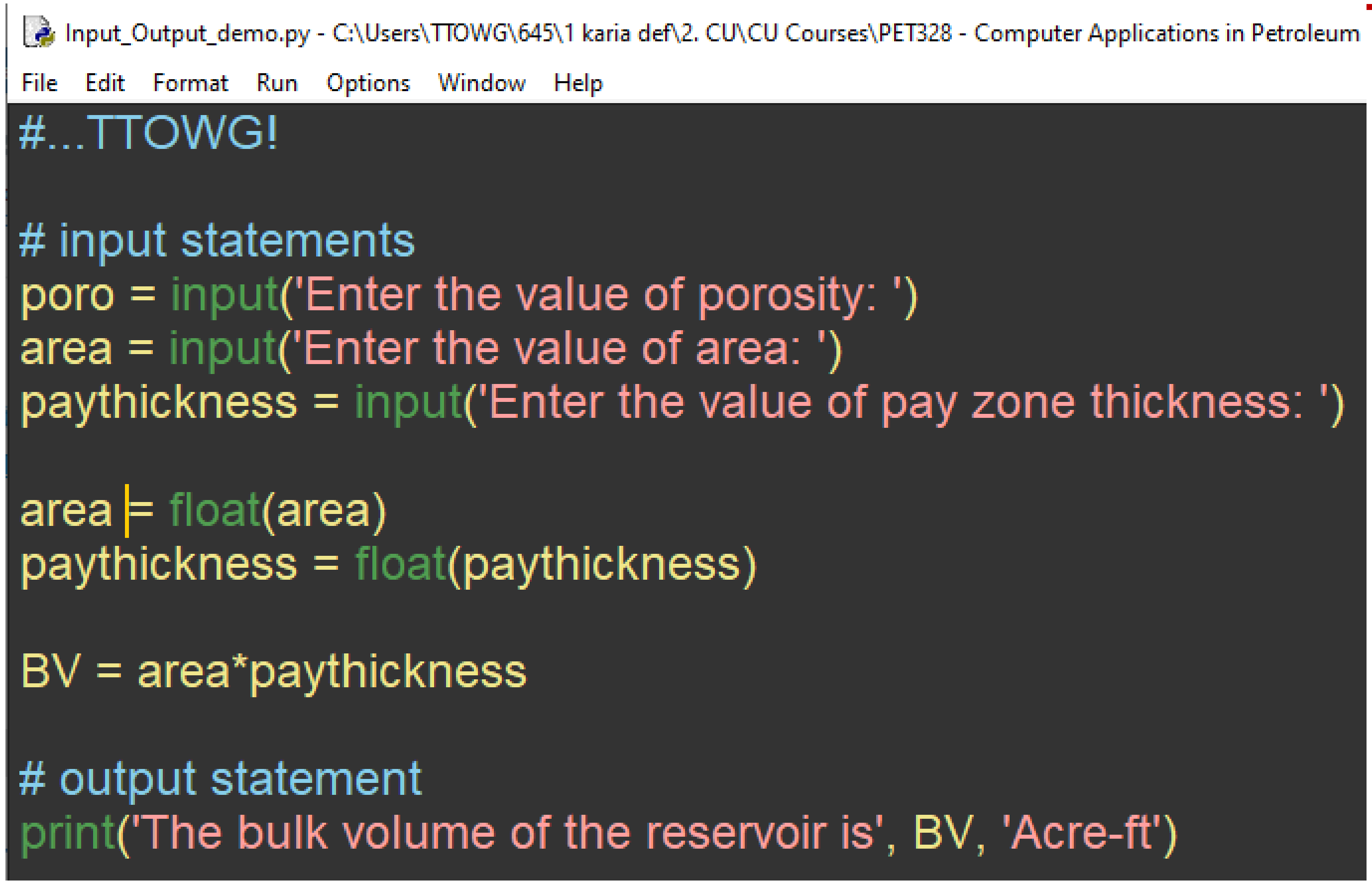
- The following are typical patterns (statement(s)) you see in a program:
  - Input statements
  - Output statements
  - Sequential execution
  - Conditional statements
  - Repeated execution (loops)
  - Reuse of statements (functions)

PREAMBLES

Introduction to Computer Programming

Fundamental patterns (concepts) in a program

- Input statements: - used to request and accept data from users
- Example: the input function.

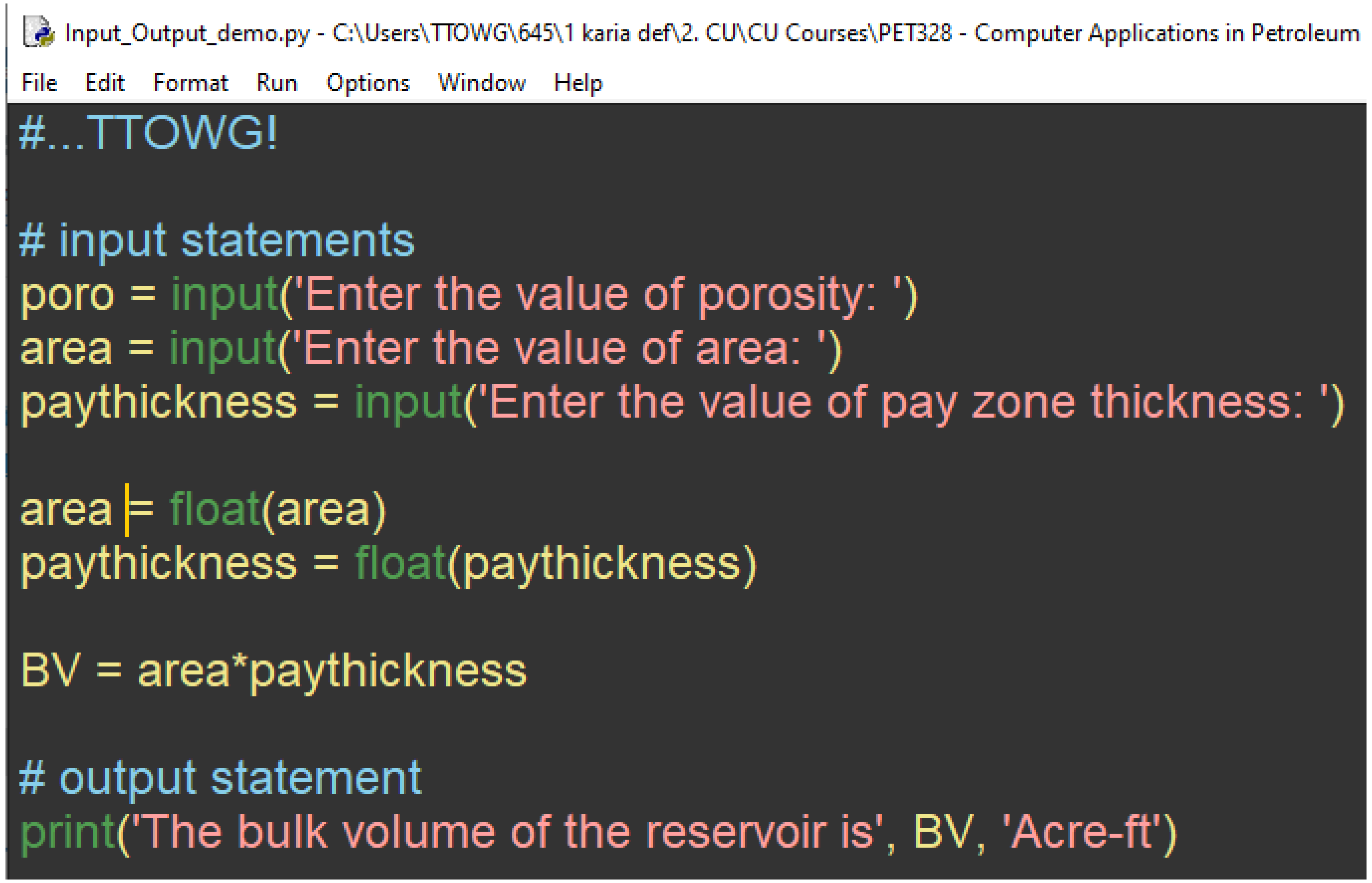

Input_Output_demo.py - C:\Users\TTOWG\645\1 karia def\2. CU\CU Courses\PET328 - Computer Applications in Petroleum
File Edit Format Run Options Window Help

```
#...TTOWG!

# input statements
poro = input('Enter the value of porosity: ')
area = input('Enter the value of area: ')
paythickness = input('Enter the value of pay zone thickness: ')

area = float(area)
paythickness = float(paythickness)

BV = area*paythickness

# output statement
print('The bulk volume of the reservoir is', BV, 'Acre-ft')
```

Introduction to Computer Programming

Fundamental patterns (concepts) in a program

- Output statements: - used to display the results of execution on the screen.
- Example: the print function



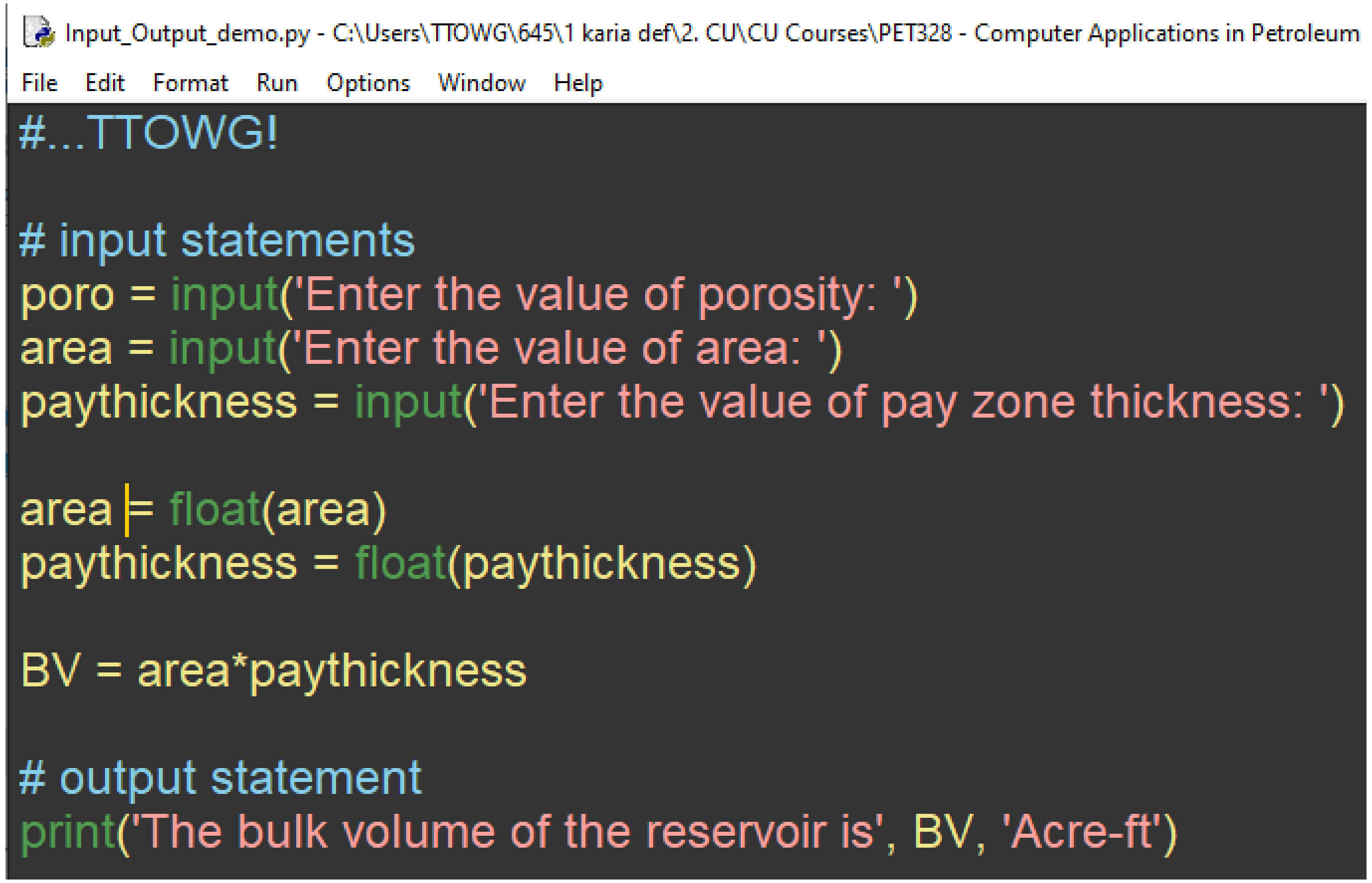
Input_Output_demo.py - C:\Users\TTOWG\645\1 karia def\2. CU\CU Courses\PET328 - Computer Applications in Petroleum

```
#...TTOWG!

# input statements
poro = input('Enter the value of porosity: ')
area = input('Enter the value of area: ')
paythickness = input('Enter the value of pay zone thickness: ')

area = float(area)
paythickness = float(paythickness)

BV = area*paythickness

# output statement
print('The bulk volume of the reservoir is', BV, 'Acre-ft')
```

Introduction to Computer Programming

Fundamental patterns (concepts) in a program

- Sequential execution: - typically, a program would entail multiple statements.
- Statements are executed in the order (sequence) in which they are encountered.
- Latter statements can make use of results of former statements; not vice-versa

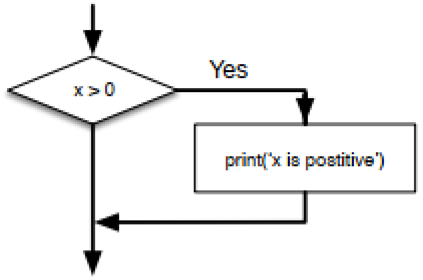

Input_Output_demo.py - C:\Users\TTOWG\645\1 karia def\2. CU\CU Courses\PET328 - Computer Applications in Petroleum

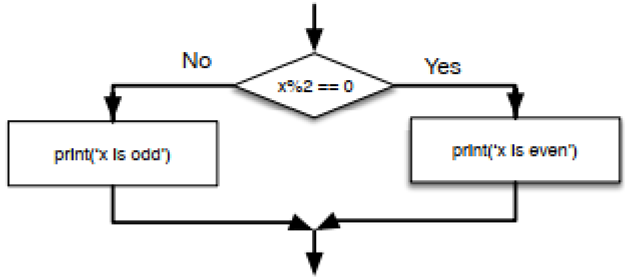
```
#...TTOWG!

# input statements
poro = input('Enter the value of porosity: ')
area = input('Enter the value of area: ')
paythickness = input('Enter the value of pay zone thickness: ')

area = float(area)
paythickness = float(paythickness)

BV = area*paythickness

# output statement
print('The bulk volume of the reservoir is', BV, 'Acre-ft')
```

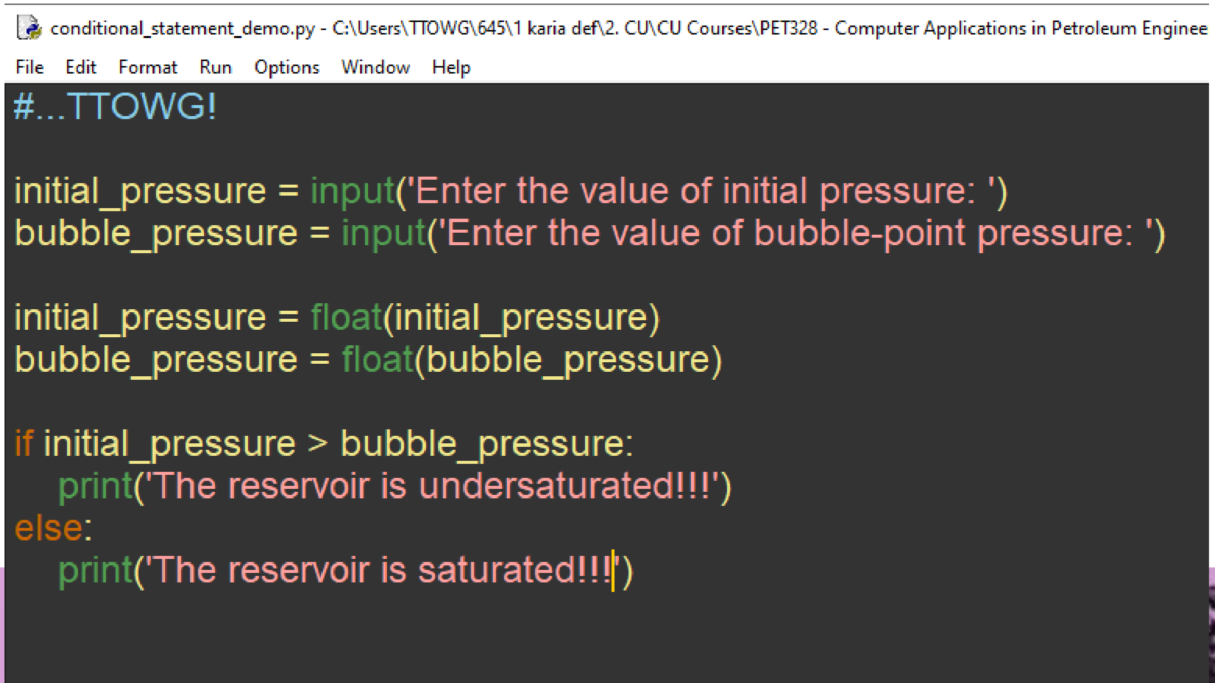
Introduction to Computer Programming

Fundamental patterns (concepts) in a program

- Conditional Statements: patterns that make it possible for the program to check for some conditions and decide to:
  - perform a statement(s) or skip the statement(s)
  - Choose between alternative statements.





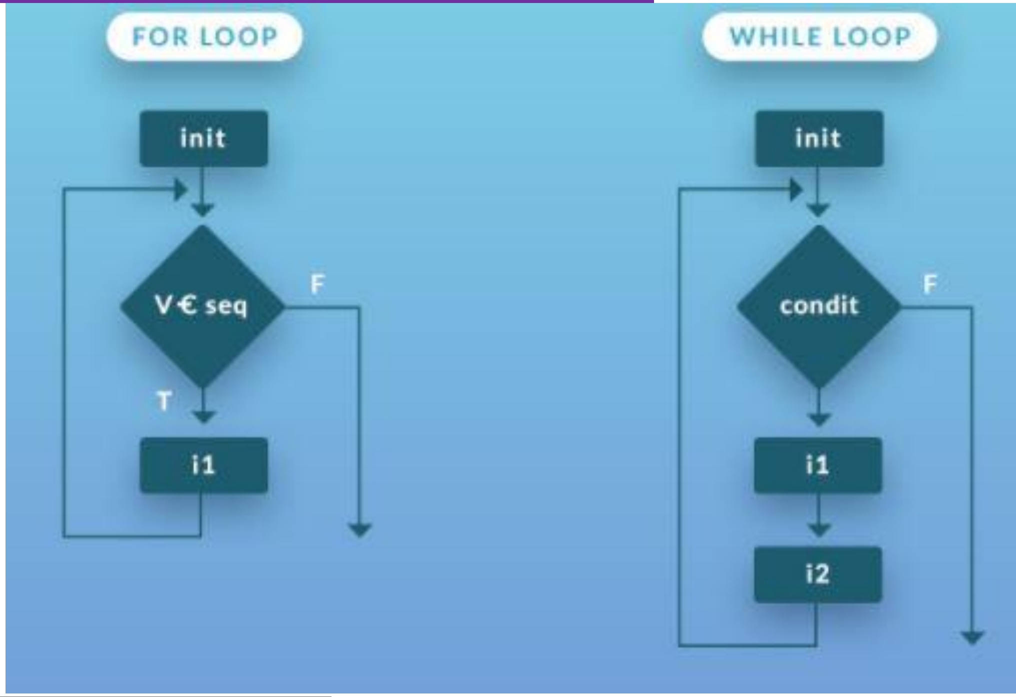

conditional_statement_demo.py - C:\Users\TTOWG\645\1 karia def\2. CU\CU Courses\PET328 - Computer Applications in Petroleum Enginee
File Edit Format Run Options Window Help

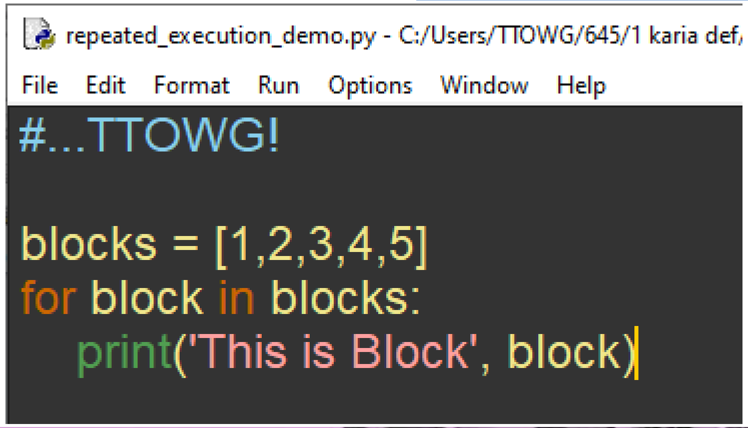
```
#...TTOWG!

initial_pressure = input('Enter the value of initial pressure: ')
bubble_pressure = input('Enter the value of bubble-point pressure: ')

initial_pressure = float(initial_pressure)
bubble_pressure = float(bubble_pressure)

if initial_pressure > bubble_pressure:
    print('The reservoir is undersaturated!!!')
else:
    print('The reservoir is saturated!!!')
```

Introduction to Computer Programming

Fundamental patterns (concepts) in a program

- Repeated Execution: patterns that instructs the program to perform (iterate) a given statement(s) repeatedly, for each item in a set of items, varying values of parameter(s) from item to item.



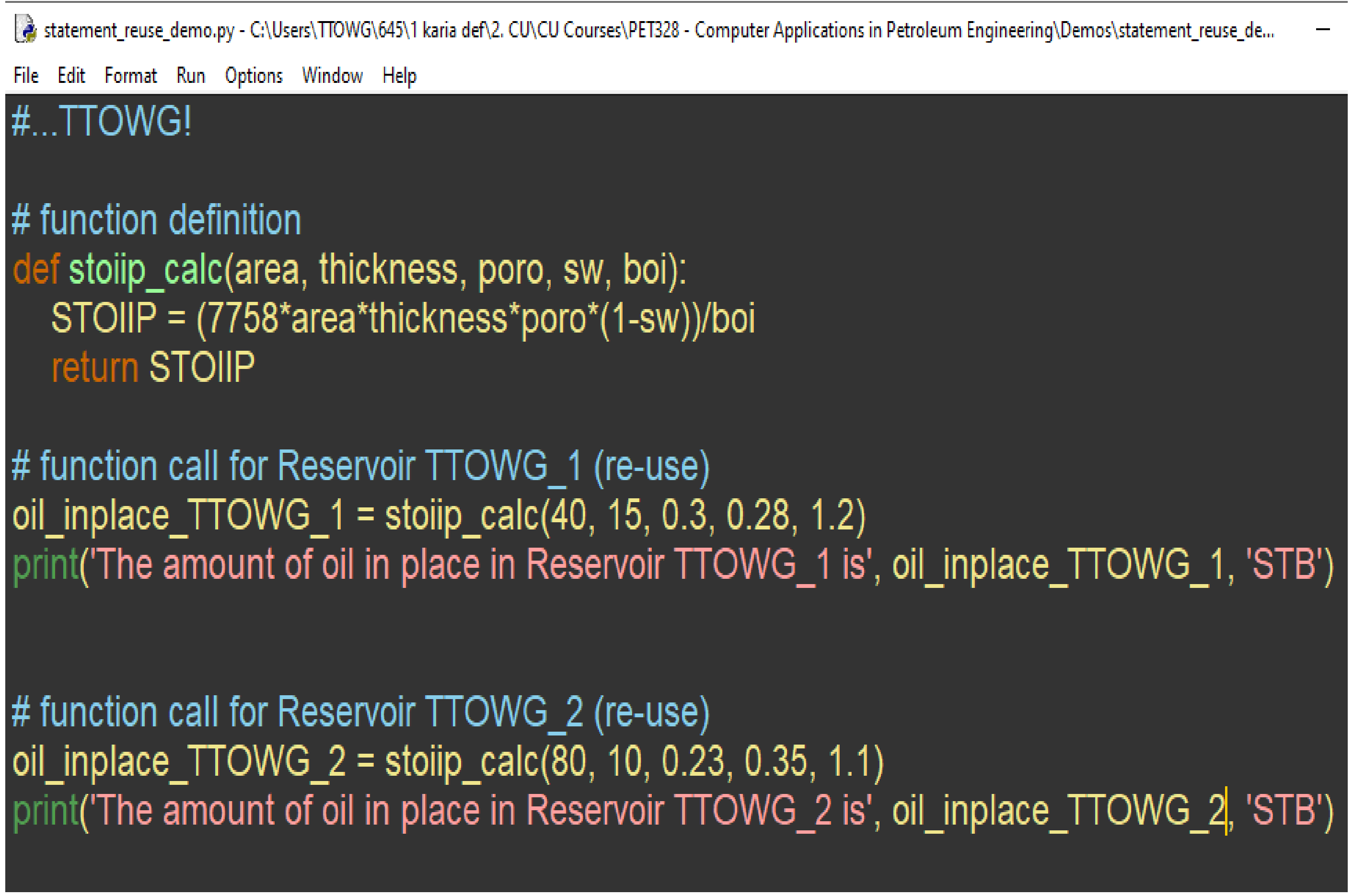

repeated_execution_demo.py - C:/Users/TTOWG/645/1 karia def/
File Edit Format Run Options Window Help

```
#...TTOWG!

blocks = [1,2,3,4,5]
for block in blocks:
    print('This is Block', block)
```

Introduction to Computer Programming

Fundamental patterns (concepts) in a program

- Re-use of statements: the task performed by some statement(s) might be routine and needed at various points in your program.
- Such statement(s) may be written once, saved with a name and re-used at various points in your program by referring to the name.


statement_reuse_demo.py - C:\Users\TTOWG\645\1 karia def\2. CU\CU Courses\PET328 - Computer Applications in Petroleum Engineering\Demos\statement_reuse_de...

```python
#...TTOWG!

# function definition
def stoiip_calc(area, thickness, poro, sw, boi):
    STOIIP = (7758*area*thickness*poro*(1-sw))/boi
    return STOIIP

# function call for Reservoir TTOWG_1 (re-use)
oil_inplace_TTOWG_1 = stoiip_calc(40, 15, 0.3, 0.28, 1.2)
print('The amount of oil in place in Reservoir TTOWG_1 is', oil_inplace_TTOWG_1, 'STB')


# function call for Reservoir TTOWG_2 (re-use)
oil_inplace_TTOWG_2 = stoiip_calc(80, 10, 0.23, 0.35, 1.1)
print('The amount of oil in place in Reservoir TTOWG_2 is', oil_inplace_TTOWG_2, 'STB')
```

```
>>>#TTOWG!
>>>print('…to the only wise God')
```

Coming Soon…

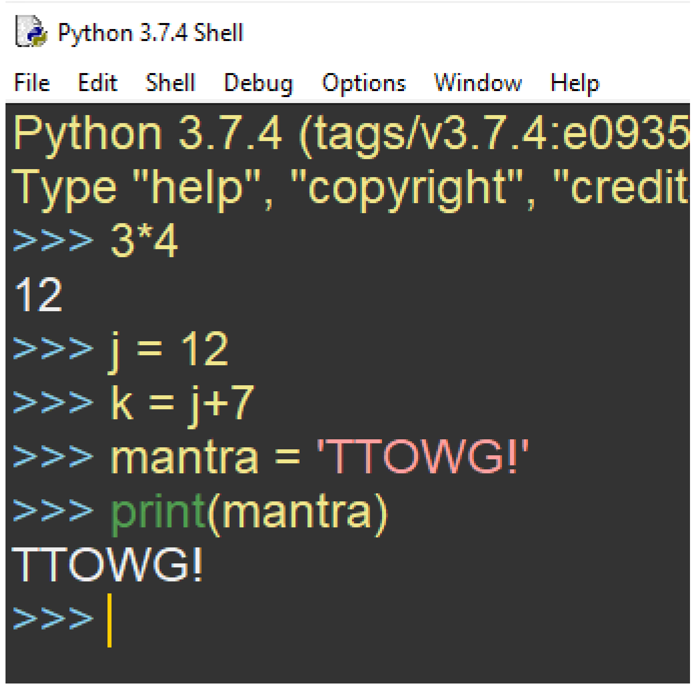
Season 1 Episode 2 –

Getting Started with Python

Basic Python Objects

- Crudely speaking, Python objects are stuffs upon which actions (specified in python commands) are performed.

- Example: in the code screenshot shown, 3, 4, j, k, 7, mantra, 'TTOWG' are all objects acted upon.



```
Python 3.7.4 (tags/v3.7.4:e0935
Type "help", "copyright", "credit
>>> 3*4
12
>>> j = 12
>>> k = j+7
>>> mantra = 'TTOWG!'
>>> print(mantra)
TTOWG!
>>> 
```

Basic Python Objects

- The basic Python objects considered here are Values and Variables.
- Later, some sets of sophisticated objects known as data structure shall be considered.

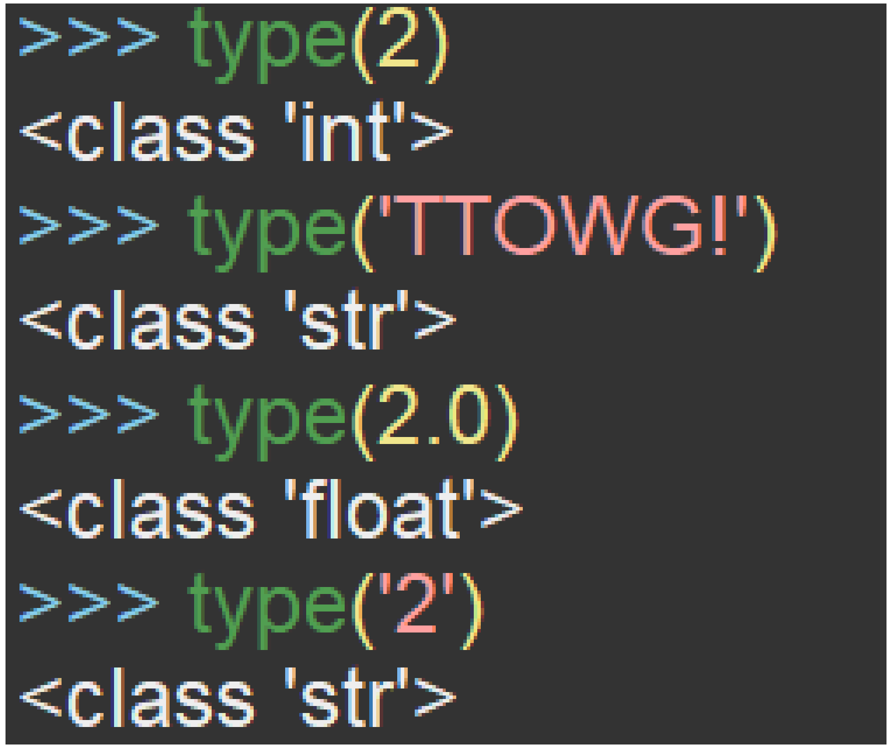
Basic Python Objects

Values

- Values are simply the representation of data entities.

Types of Values

- Values in Python belong to various types such as type *integer*, type *float*, and type *string*.
- Use the function *type* to find out the type to which a value belong.

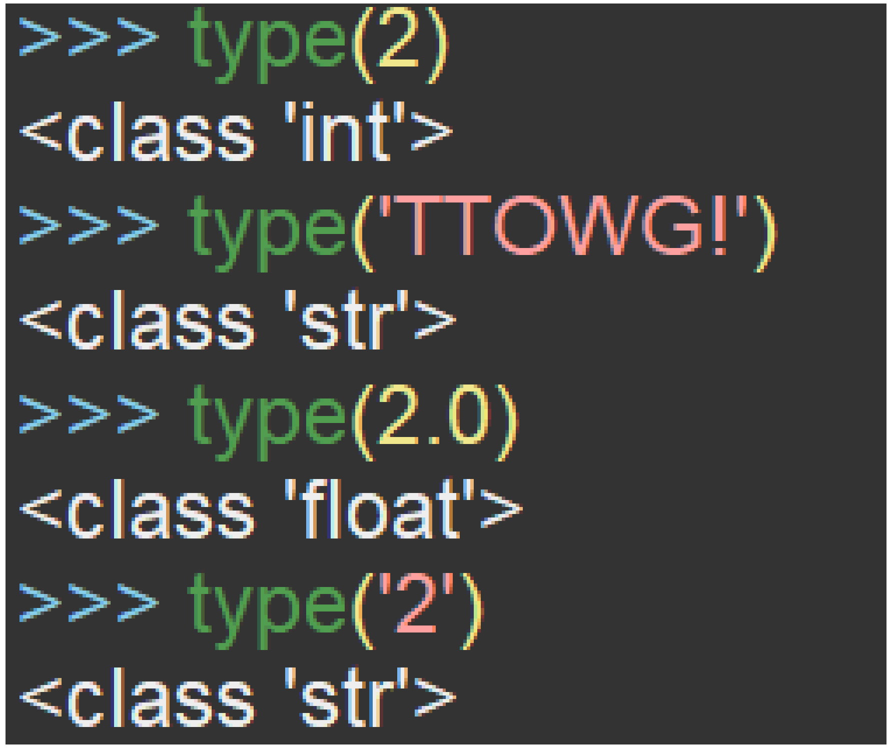


```
>>> type(2)
<class 'int'>
>>> type('TTOWG!')
<class 'str'>
>>> type(2.0)
<class 'float'>
>>> type('2')
<class 'str'>
```

Basic Python Objects

Types of Values

- 2 is of type (class) integer
- 'TTOWG' and '2' are of type string; just like any set of characters (alphanumeric and non-alphanumeric) enclosed in quotes
- 2.0 is of type float; just as are all numbers expressed in decimals.



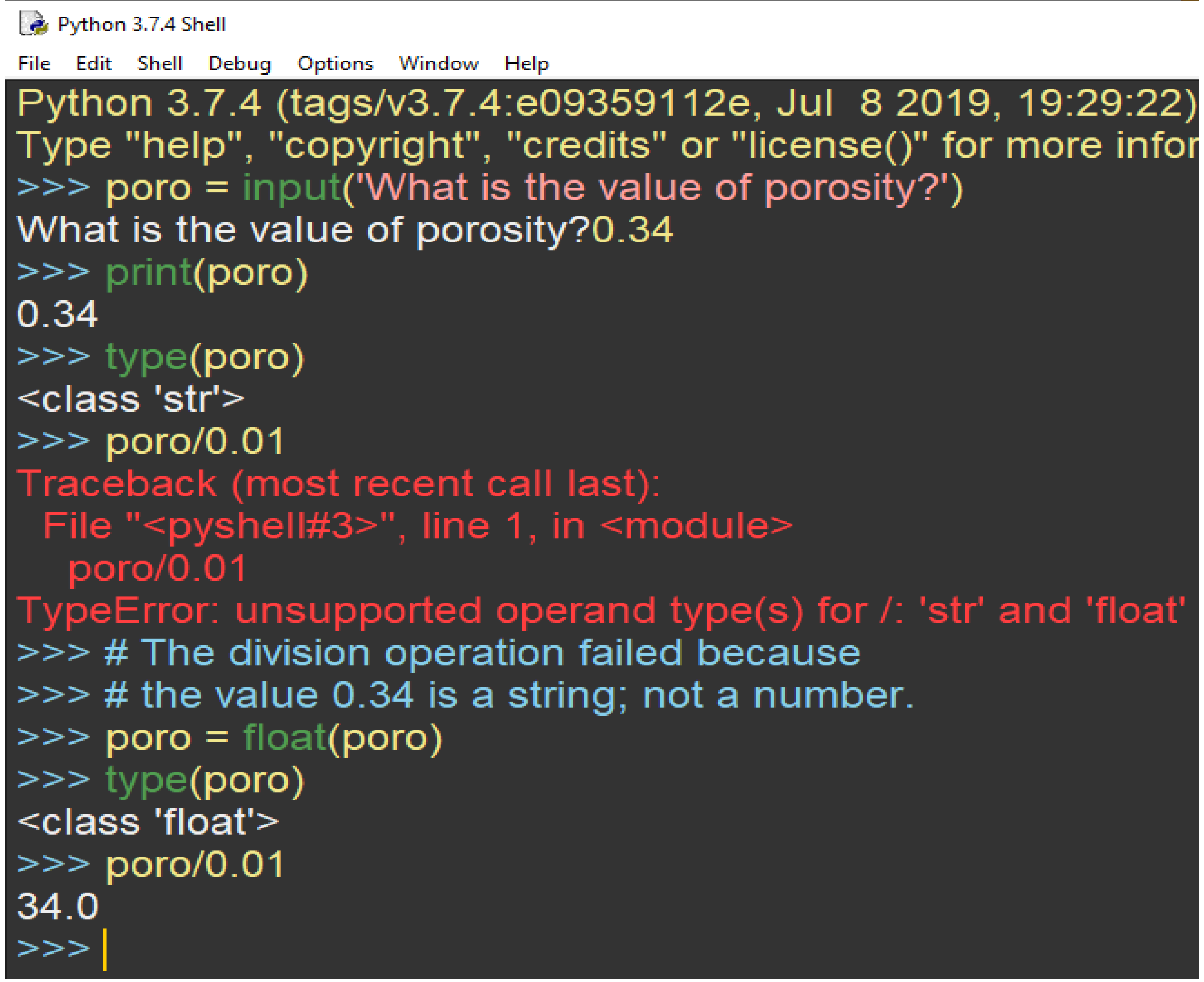
```
>>> type(2)
<class 'int'>
>>> type('TTOWG!')
<class 'str'>
>>> type(2.0)
<class 'float'>
>>> type('2')
<class 'str'>
```

GETTING STARTED WITH PYTHON

Basic Python Objects

Types of Values

- Please, take note that users' response to the input function prompt is stored as a string.
- Before using such *input* in a mathematical operation, they should be converted to a numerical type using *float* or *int* functions.



```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22)
Type "help", "copyright", "credits" or "license()" for more infor
>>> poro = input('What is the value of porosity?')
What is the value of porosity?0.34
>>> print(poro)
0.34
>>> type(poro)
<class 'str'>
>>> poro/0.01
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    poro/0.01
TypeError: unsupported operand type(s) for /: 'str' and 'float'
>>> # The division operation failed because
>>> # the value 0.34 is a string; not a number.
>>> poro = float(poro)
>>> type(poro)
<class 'float'>
>>> poro/0.01
34.0
>>> 
```

Basic Python Objects

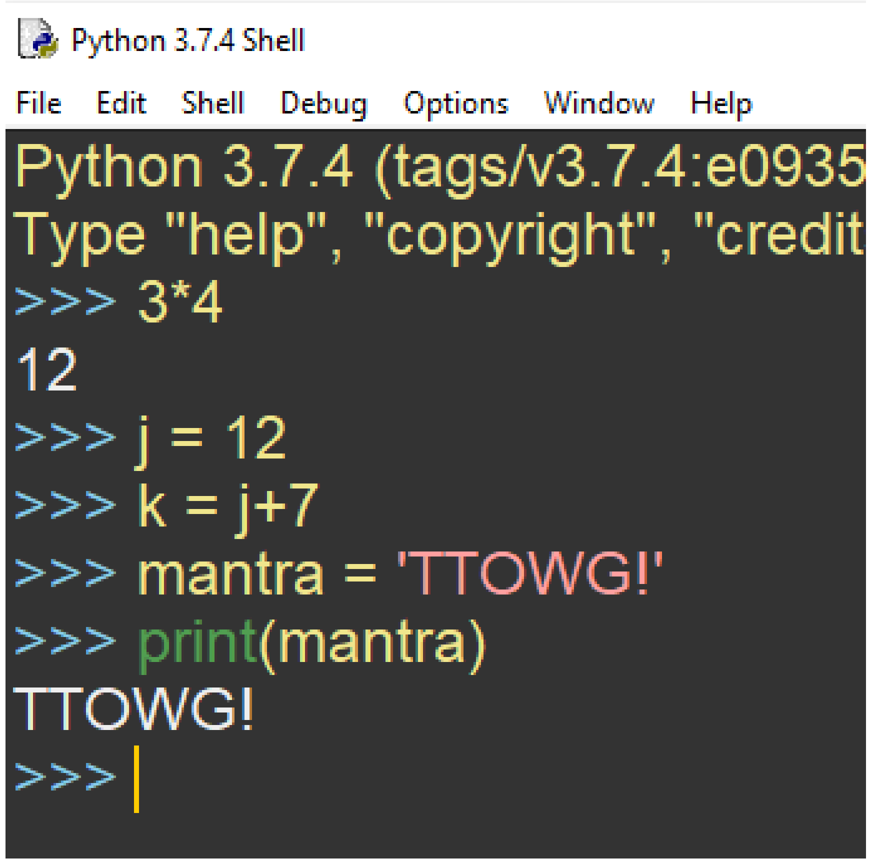
Variables

- A variable is a value stored in memory and referred to with a chosen name.
- In other words, values are assigned to variables.
- When the name of a variable is called, the value assigned therein answers.

Basic Python Objects

Variables

- A raw value can be assigned to a variable.
    - Example: j is a variable; the value 12 is assigned to it.
- Also, the output of an expression (involving a variable) may be stored in another variable.
    - Example: k is a variable, the value obtained when j+7 is executed is subsequently assigned to variable k.
- Not only numeric values are assigned to variables, strings are also assigned.
    - Example, mantra is a variable with string ‘TTOWG!’ assigned to it.



```
Python 3.7.4 (tags/v3.7.4:e0935
Type "help", "copyright", "credit
>>> 3*4
12
>>> j = 12
>>> k = j+7
>>> mantra = 'TTOWG!'
>>> print(mantra)
TTOWG!
>>> 
```

Basic Python Objects

Choosing Variable Names

- In naming variable, the following rules are recommended:
  - Variable names should be descriptive, as much as possible. That is, the name should somewhat tell us something about the variable. Example: a variable to hold the value of reservoir permeability is better named 'perm' than named 'x'
  - The name must be a single word. Where multiple words are necessary for descriptive purposes, they can be joined with the underscore character; e. g.: init_pressure.
  - Names should not be too long.
  - Names may contain both alphabets and numbers; but must not start with numbers.
  - Names are case sensitive. If you named a variable as 'poro', do not refer to it as 'Poro'.
  - Avoid using special characters like '@', '$' in names.

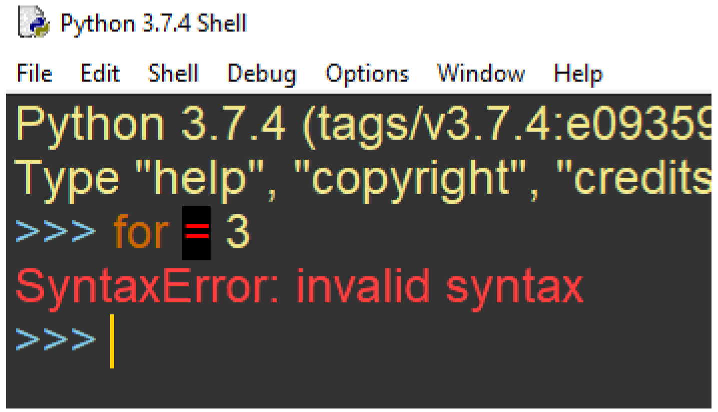
GETTING STARTED WITH PYTHON

Basic Python Objects

Keywords

- Keywords are words that are reserved for Python's in-built structure.
- Here is the list of Python's keywords.
- Keywords cannot be used as variable names; doing so would cause error.

and	del	from	None	True
as	elif	global	nonlocal	try
assert	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield
continue	finally	is	raise	async
def	for	lambda	return	await



```
Python 3.7.4 (tags/v3.7.4:e0935
Type "help", "copyright", "credits
>>> for = 3
SyntaxError: invalid syntax
>>> 
```

Basic Python Objects

Statements

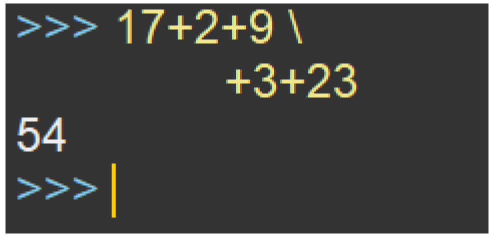
- A statement is simply unit of code (commands) that is interpretable and executable by Python; just like a sentence in natural language.
- Two common types of Python statements are Assignment statements and Expressions.
- Assignment statements simply assigns values to a variable.
- An expression is a statement that combines variables, values, functions and operators.
- A statement could combine both types such that the result of an expression (RHS) is assigned to a variable (LHS).

```
poro = 0.27 # This is an assignment statement.

area = 40 # This is an assignment statement.

thickness = 15 # This is an assignment statement.

area*thickness # This is an expression.

PV = area*thickness*poro # A combination of expression (RHS) and assignment
```

Basic Python Objects

Multi-line statements

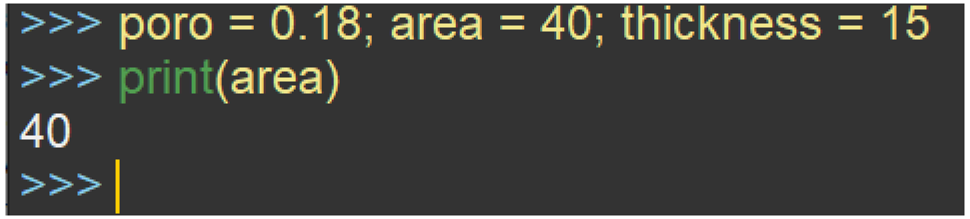
- Typically, a Python statement is written in a single line.
- However, if the statement is too long, it could be continued in the next line; but the current line should end with the line continuation character i.e. \



```
>>> 17+2+9 \
          +3+23
54
>>>
```

Multiple statements in a line

- Writing multiple statements in same line is not encouraged; however, if that has to be done, the statements should be separated by semicolon.

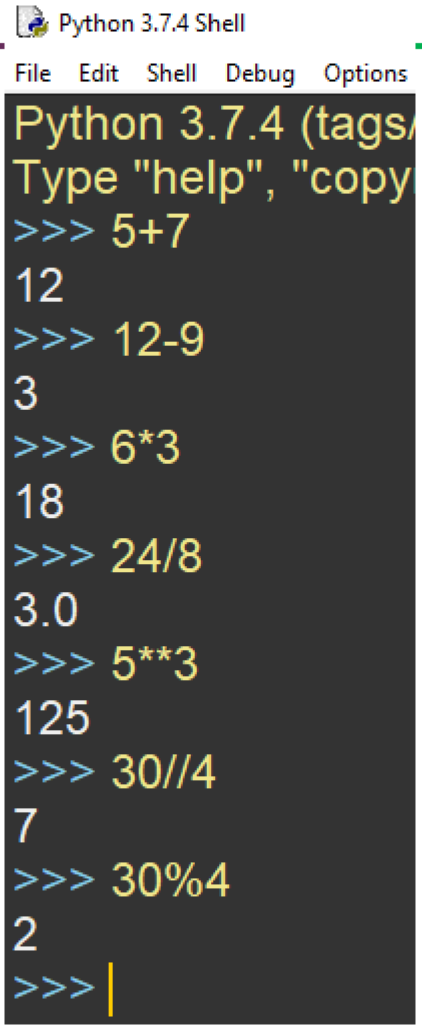


```
>>> poro = 0.18; area = 40; thickness = 15
>>> print(area)
40
>>>
```

Basic Python Objects

Operators

- Operators are symbols of mathematical operations.
  - + for addition
  - - for subtraction
  - * for multiplication
  - / for division
  - ** for exponentiation (raise to power)
  - // integer division (truncates the result of division to its integer part.
  - % modulus (gives the remainder of an integer division).
- The objects acted upon by operators are called operands.



```
Python 3.7.4 (tags/
Type "help", "copy
>>> 5+7
12
>>> 12-9
3
>>> 6*3
18
>>> 24/8
3.0
>>> 5**3
125
>>> 30//4
7
>>> 30%4
2
>>>
```

Basic Python Objects

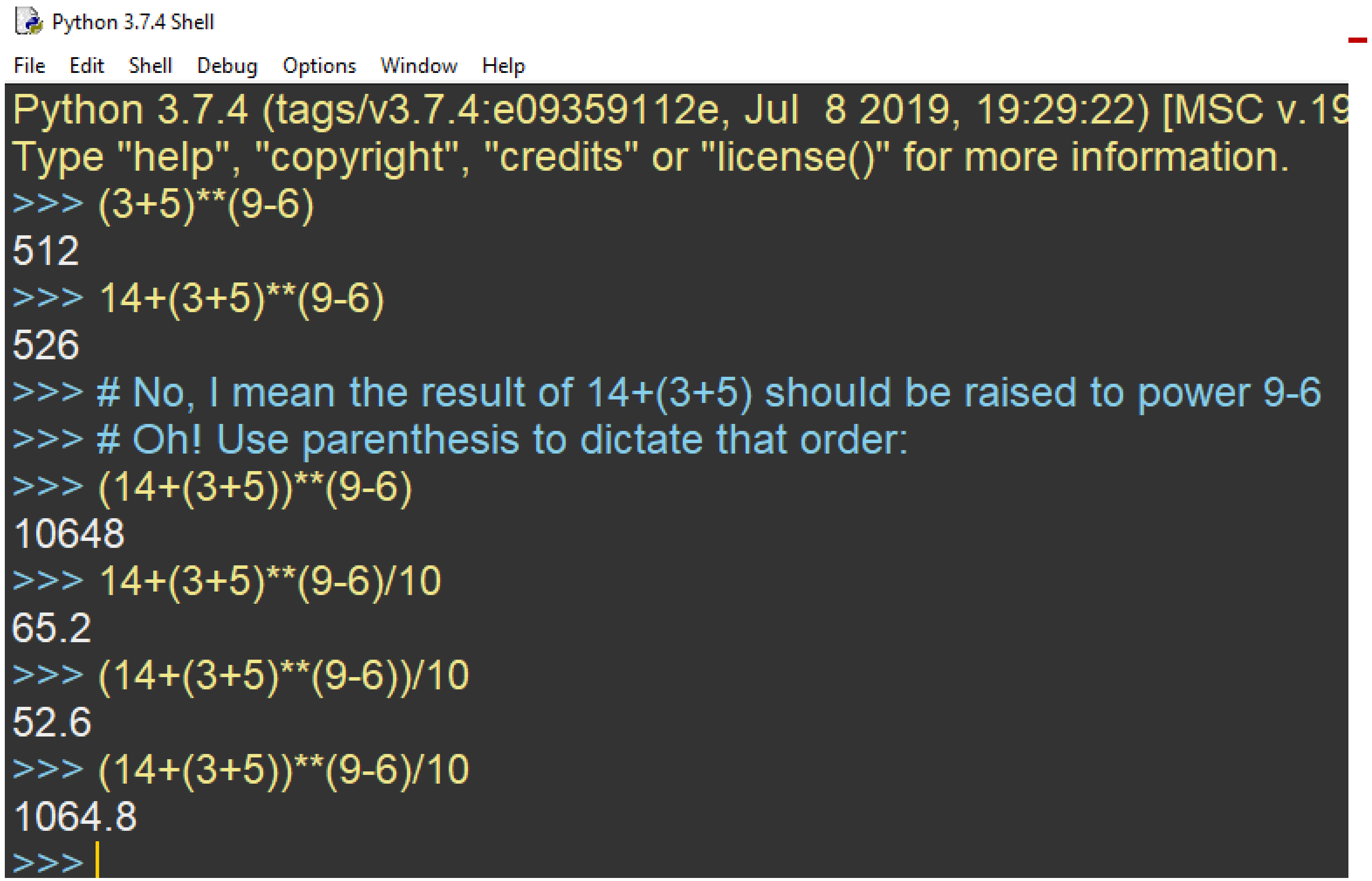
Order of Operations

- When multiple operations are featured in a statement, Python executes them in the order specified by the accronym: PE-MD-AS (Parenthesis, Exponentiation, Multiplication, Division, and Subtraction).
- You can used parenthesis to dictate the order you desire.
- Multiplication and Division has equal precedence; hence are executed left to right.
- Addittion and Subtraction has equal precedence; hence are executed left to right.
- You may also use parenthesis to make an expression more readable and less confusing.
- Nested parenthesis are executed from inside to outside.

Basic Python Objects

Order of Operations

- Consider the following operations to convince yourself of the PEMDAS order.

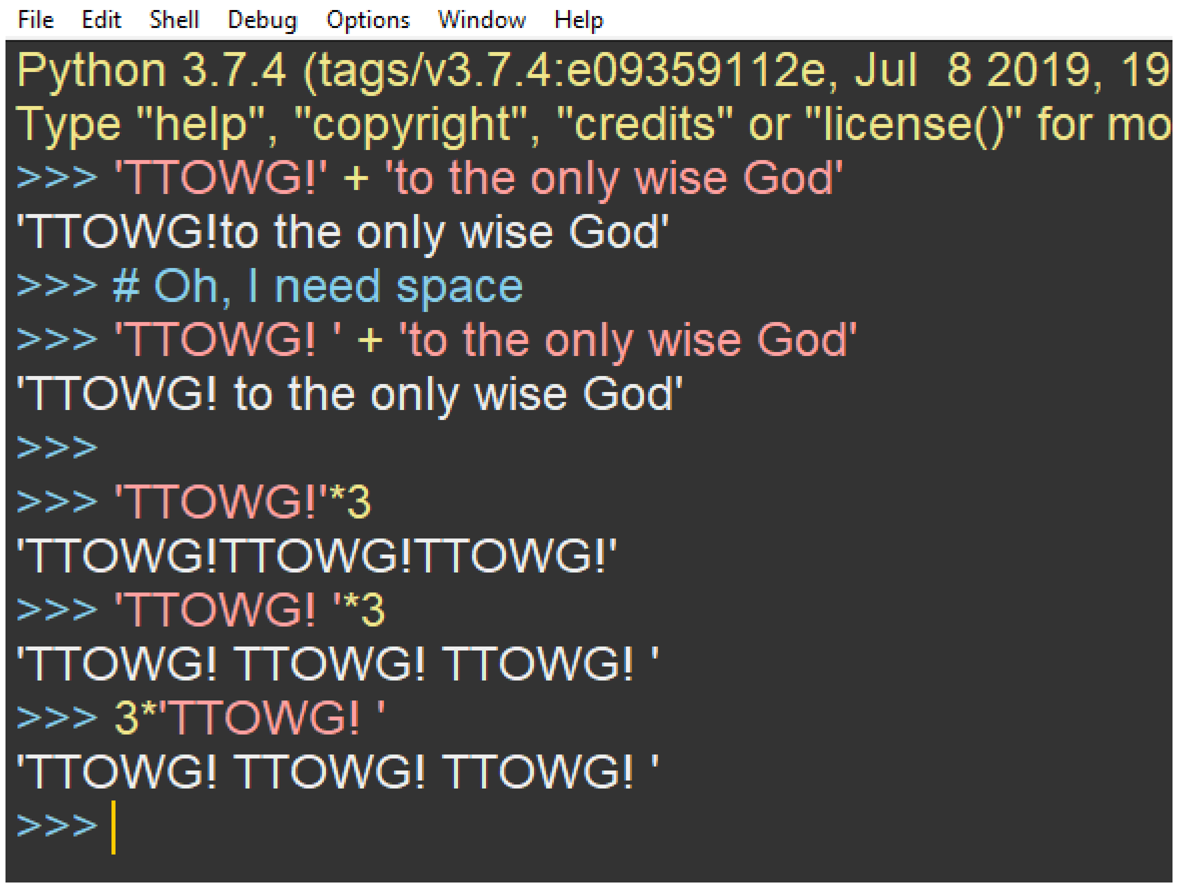


```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.19
Type "help", "copyright", "credits" or "license()" for more information.
>>> (3+5)**(9-6)
512
>>> 14+(3+5)**(9-6)
526
>>> # No, I mean the result of 14+(3+5) should be raised to power 9-6
>>> # Oh! Use parenthesis to dictate that order:
>>> (14+(3+5))**(9-6)
10648
>>> 14+(3+5)**(9-6)/10
65.2
>>> (14+(3+5)**(9-6))/10
52.6
>>> (14+(3+5))**(9-6)/10
1064.8
>>> 
```

Basic Python Objects

String Operations

- Strings can be joined end-to-end by using the + operator. If you want a space between the strings, then include it in one of the strings.
- Also, a string can be repeated multiple times using the * operator (with and integer, of course).


File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19
Type "help", "copyright", "credits" or "license()" for mo
>>> 'TTOWG!' + 'to the only wise God'
'TTOWG!to the only wise God'
>>> # Oh, I need space
>>> 'TTOWG! ' + 'to the only wise God'
'TTOWG! to the only wise God'
>>>
>>> 'TTOWG!'*3
'TTOWG!TTOWG!TTOWG!'
>>> 'TTOWG! '*3
'TTOWG! TTOWG! TTOWG! '
>>> 3*'TTOWG! '
'TTOWG! TTOWG! TTOWG! '
>>> 
```

Conditional Statements

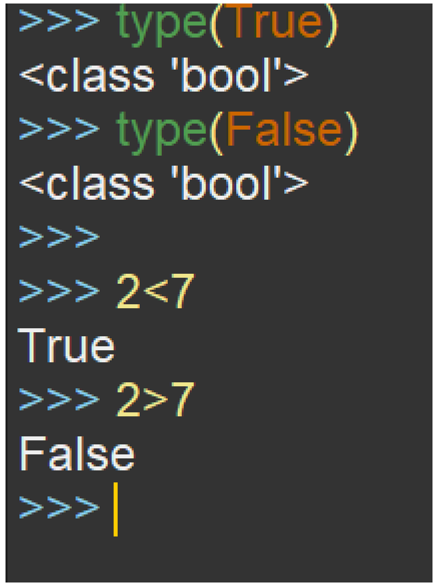
- Conditional statements are written to make it possible for a program to check for some conditions and decide to:
  - perform a statement(s) or skip the statement(s)
  - Choose between alternative statements.
- So, the concept of condition is central to this kind of statements.
- These conditions are crafted using the concept of Boolean expressions.

GETTING STARTED WITH PYTHON

Conditional Statements

Boolean Expressions

- A boolean is a value that is either True or False
- Just like the integer type can take values 1, 2, 3 e.t.c; the boolean type can take one of just two values: True or False.
- For this reason, 'True' and 'False' are Python keywords reserved for boolean values; a variable must not be named using these words.
- Now, a boolean expression is essentially a comparison expression that evaluates to either True or False.



```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>>
>>> 2<7
True
>>> 2>7
False
>>> |
```

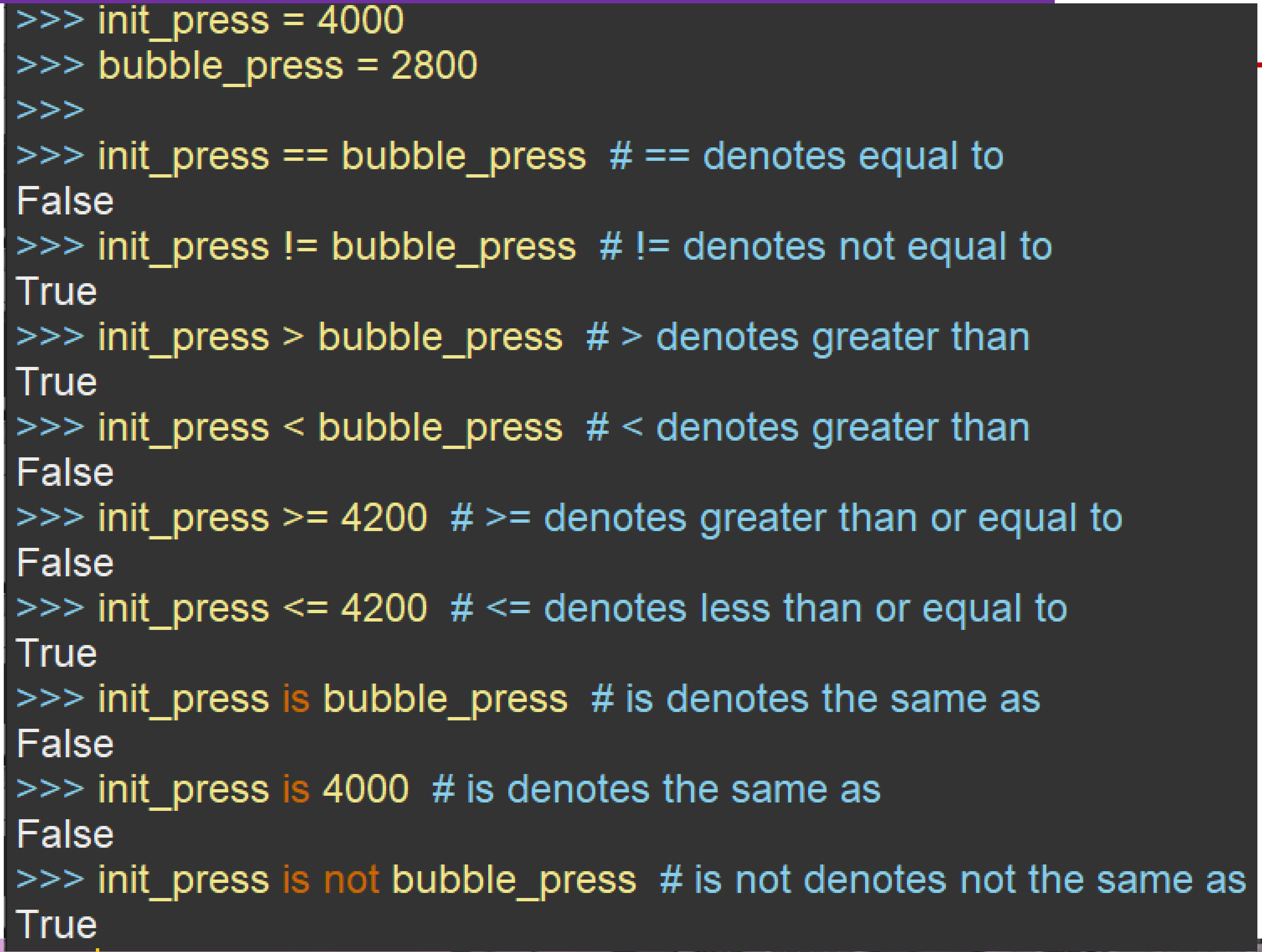
GETTING STARTED WITH PYTHON

Conditional Statements

Boolean Expressions

- Boolean expressions are constructed using comparison operators listed here.

- Take note that = is an assignment operator while == is a comparison operator.



```
>>> init_press = 4000
>>> bubble_press = 2800
>>>
>>> init_press == bubble_press  # == denotes equal to
False
>>> init_press != bubble_press  # != denotes not equal to
True
>>> init_press > bubble_press  # > denotes greater than
True
>>> init_press < bubble_press  # < denotes greater than
False
>>> init_press >= 4200  # >= denotes greater than or equal to
False
>>> init_press <= 4200  # <= denotes less than or equal to
True
>>> init_press is bubble_press  # is denotes the same as
False
>>> init_press is 4000  # is denotes the same as
False
>>> init_press is not bubble_press  # is not denotes not the same as
True
```

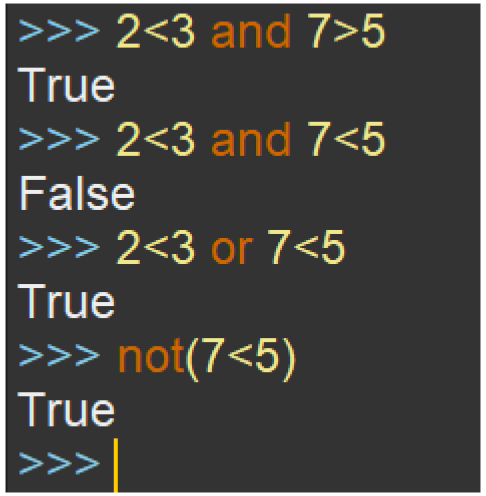
Conditional Statements

Logical Operators

- Sometimes, multiple conditions needed to be checked in a conditional statement.

- Logical operators are used to combine boolean expressions
  - *and* returns True if all conditions are true, true, otherwise, False is returned.
  - *or* returns True if one of the conditions is true, otherwise, False is returned.

- Logical operators are also used to negate boolean expressions
  - *not* returns True for a false condition and vice-versa

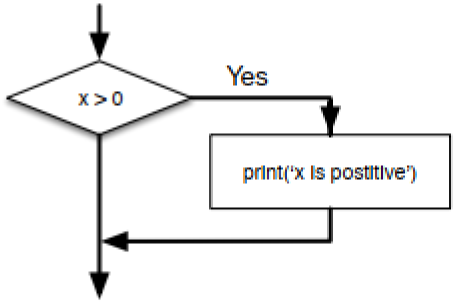


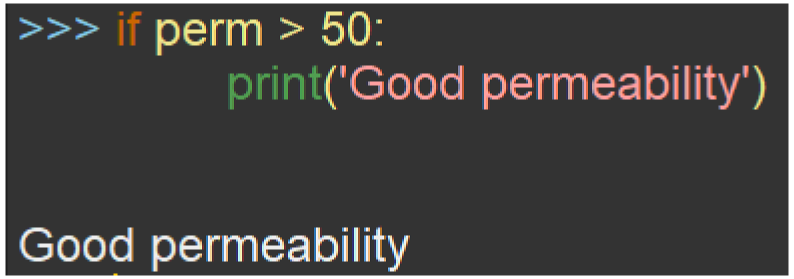
```
>>> 2<3 and 7>5
True
>>> 2<3 and 7<5
False
>>> 2<3 or 7<5
True
>>> not(7<5)
True
>>>
```

Conditional Statements

if statement

- if statements evaluates the given condition; performs the given statement(s) if condition is true and skips the given statement(s) if condition is false.
- The condition(s) is written after the *if* keyword and ended with a colon i.e. (:)
- The statements to be performed or skip are written as an indented block in subsequent line(s).
- Remove the indentation in lines after the if block.





```
>>> if perm > 50:
        print('Good permeability')


Good permeability
```