Covenant University

Raising a new Generation of Leaders

PET328

COMPUTER APPLICATIONS IN PETROLEUM ENGINEERING

PET328: COMPUTER APPLICATIONS IN PETROLEUM ENGINEERING

(With Python Programming)

python Programming

Olatunde O. Mosobalaje (PhD)

Department of Petroleum Engineering,
Covenant University, Ota
Nigeria

OUTLINE

- Preambles
  - The Appetizer
  - The Toolbox
  - The Embedded Course
  - Introduction to Computer Programming
- Getting Started with Python
  - Basic Python Objects
  - Conditional Execution
  - Repeated Execution
  - Functions
- Python Data Structures
  - Strings
  - Lists
  - Tuples
  - Dictionaries
- Some Python Libraries
  - NumPy
  - Matplotlib
  - Pandas
  - Scikit-learn
- Application Projects
  - Oil Reservoir Volumetrics
  - Material Balance Analysis
  - PVT Properties

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/

PREAMBLES

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

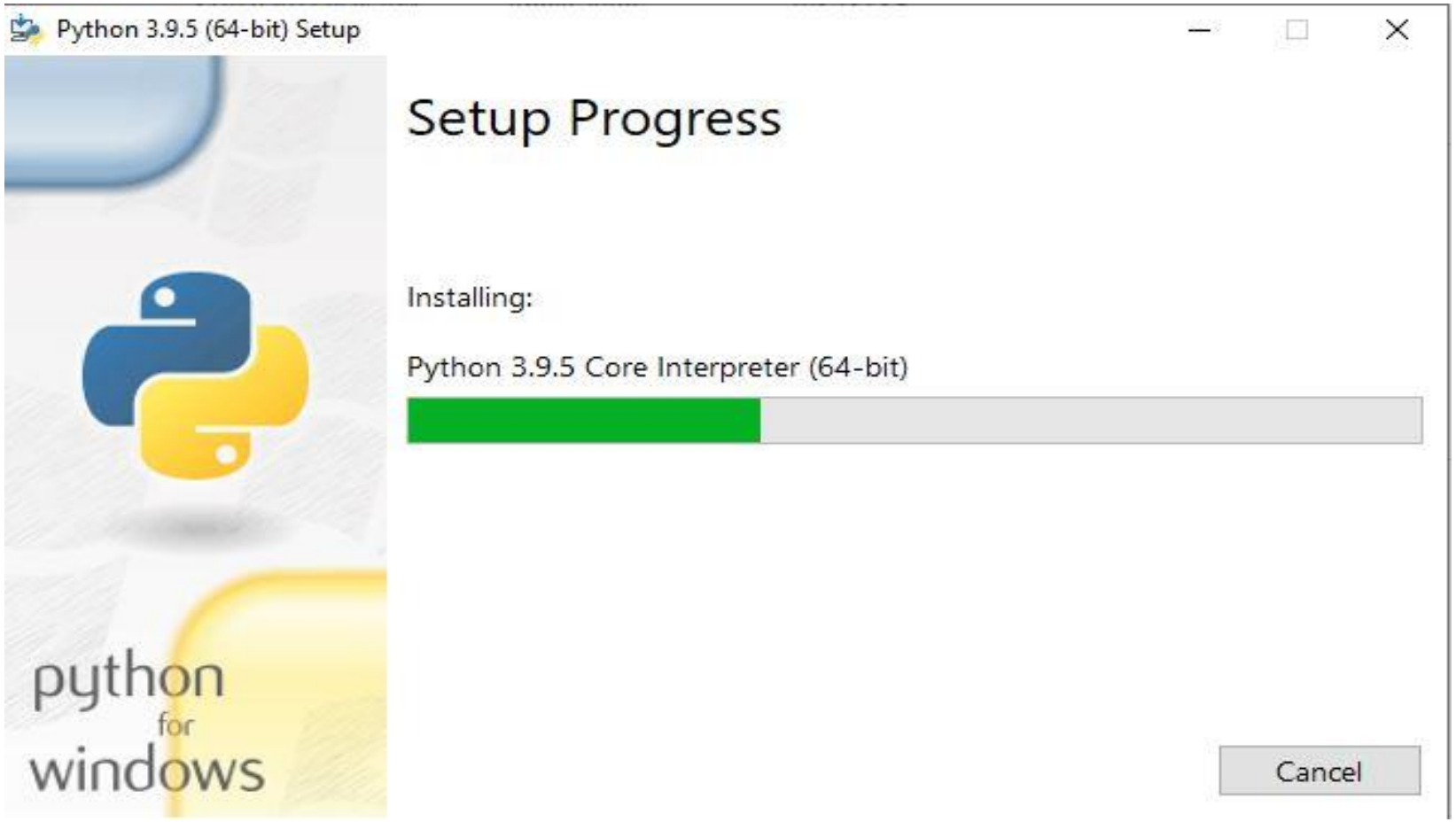
Click on the default installation option.

Ensure to check the Add Python 3.9 to PATH

PREAMBLES

The Toolbox

- Installing Python 3

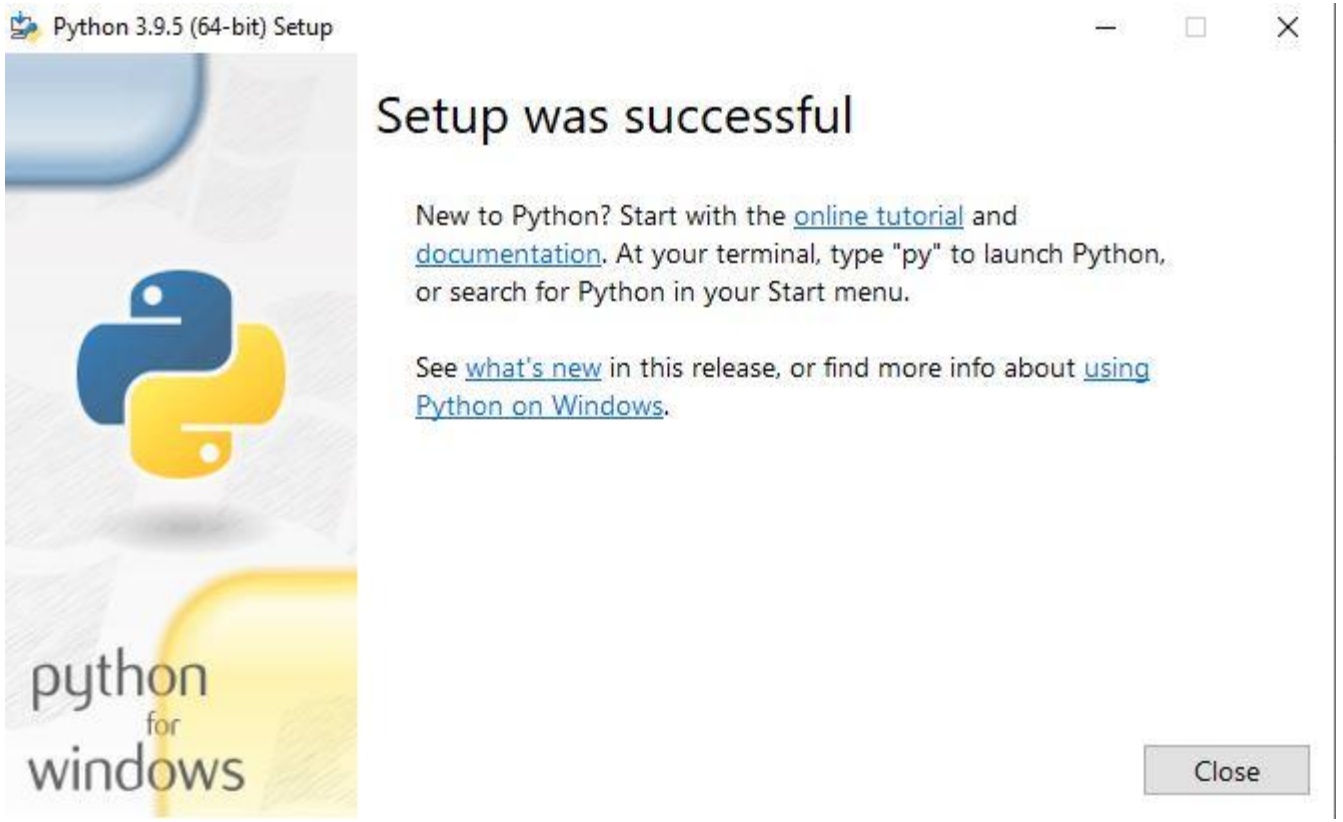


PREAMBLES

The Toolbox

- Installing Python 3

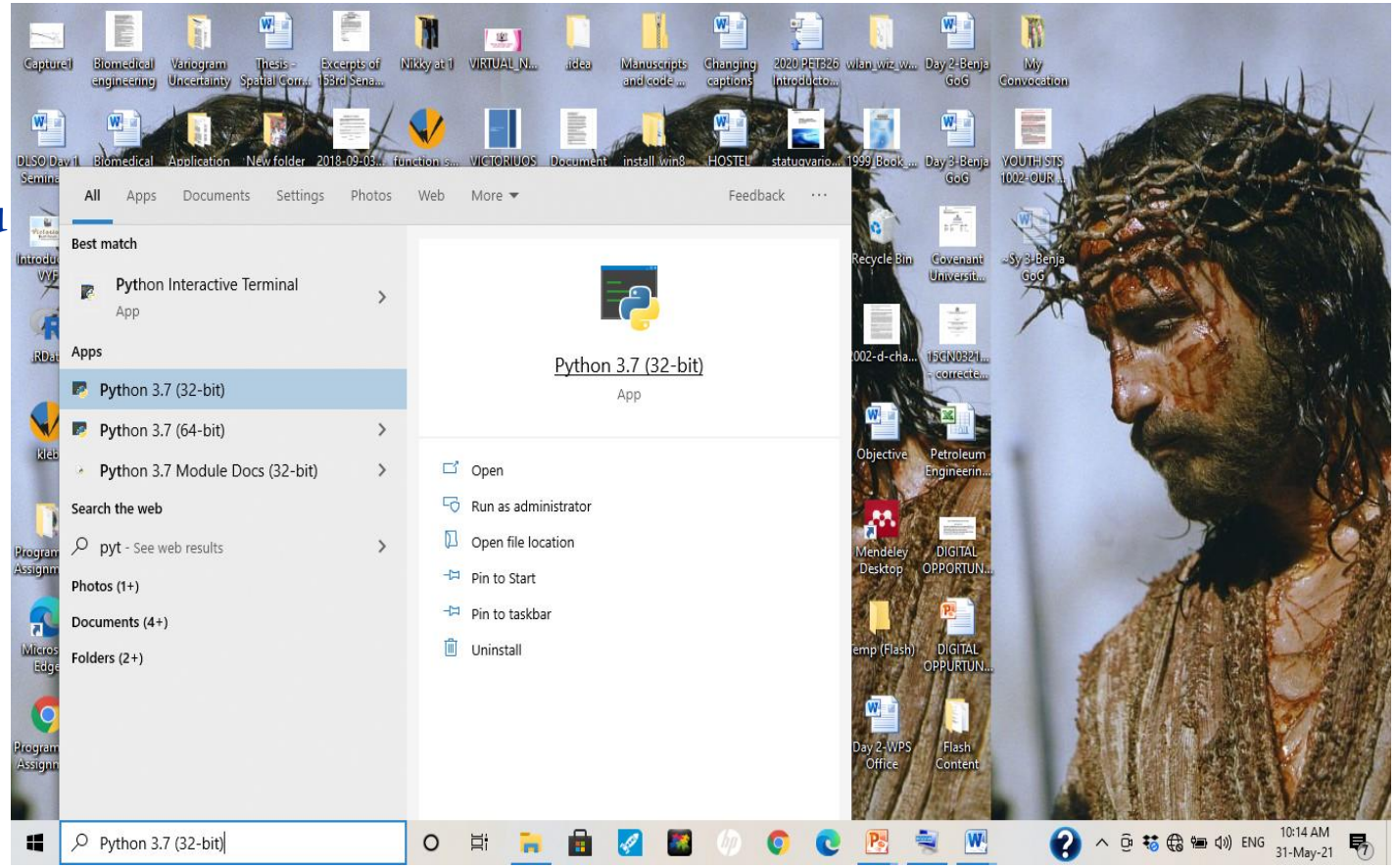
Click the close button when the installation is completed



The Toolbox

Launching Python 3

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

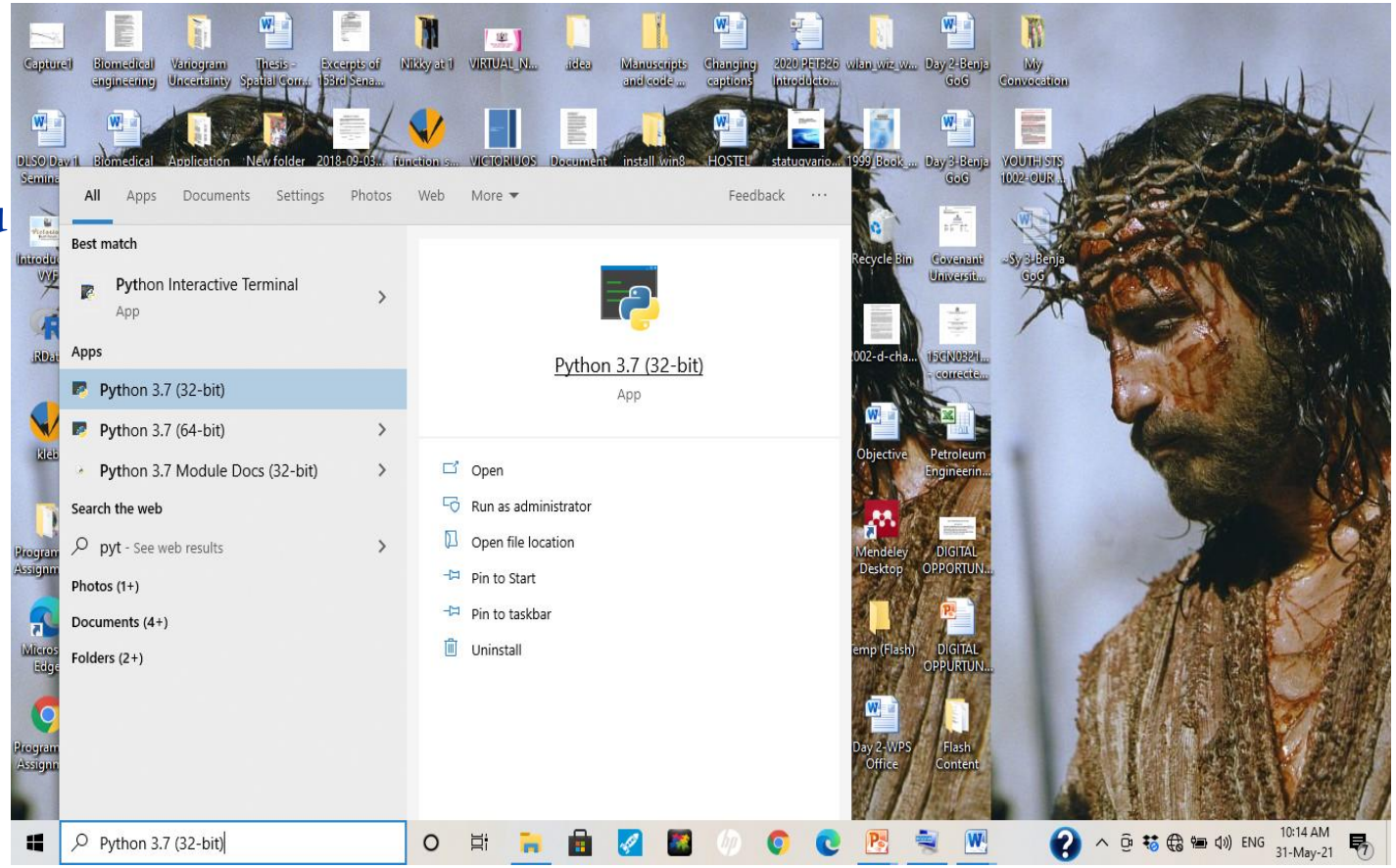


PREAMBLES

The Toolbox

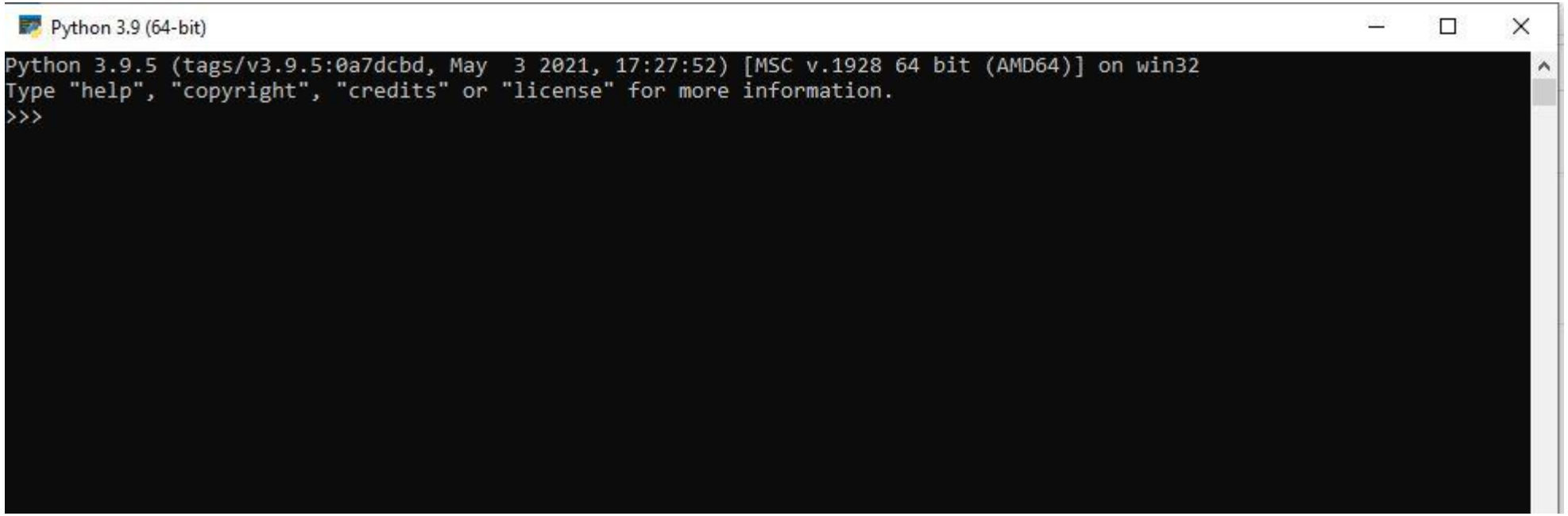
- Launching Python 3

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



The Toolbox

- Launching Python 3



The Toolbox

- Python IDLE

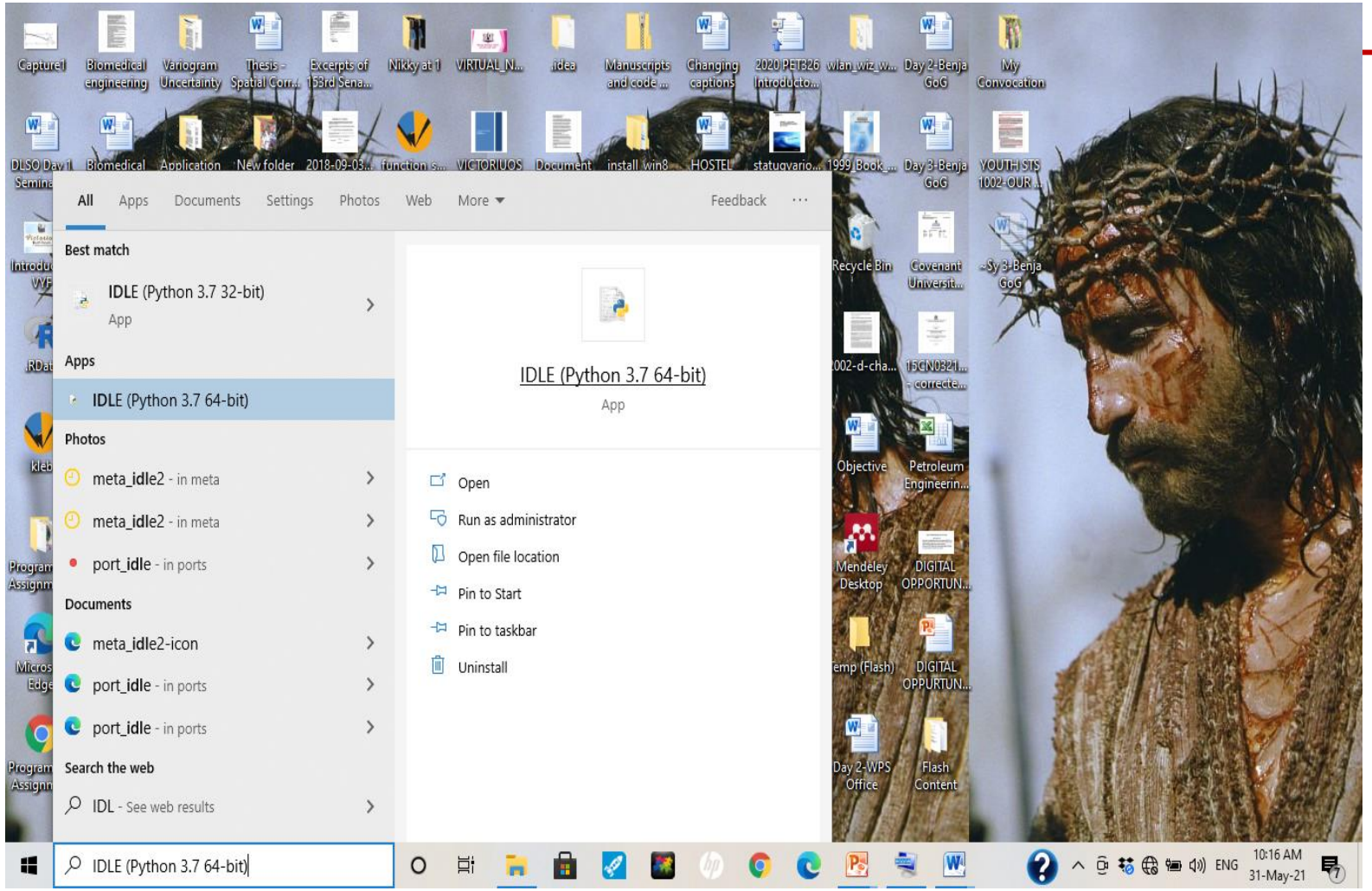
Now, the Python DOS-like environment seems boring. Good enough, we will typically not be working on that platform; rather we will interact with Python from a platform known as Interactive Development and Learning Environment (IDLE)

PREAMBLES

The Toolbox

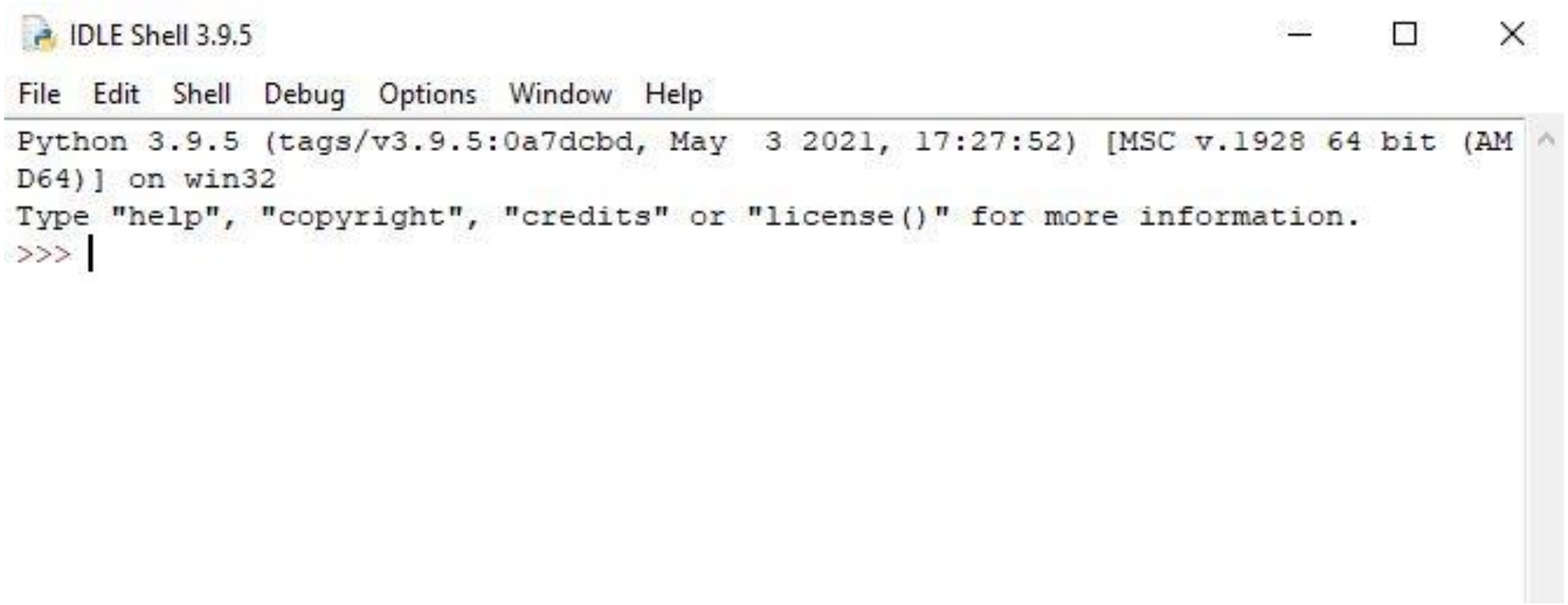
- Python IDLE

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



The Toolbox

- Python IDLE



PREAMBLES

The Toolbox

- Python IDLE

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

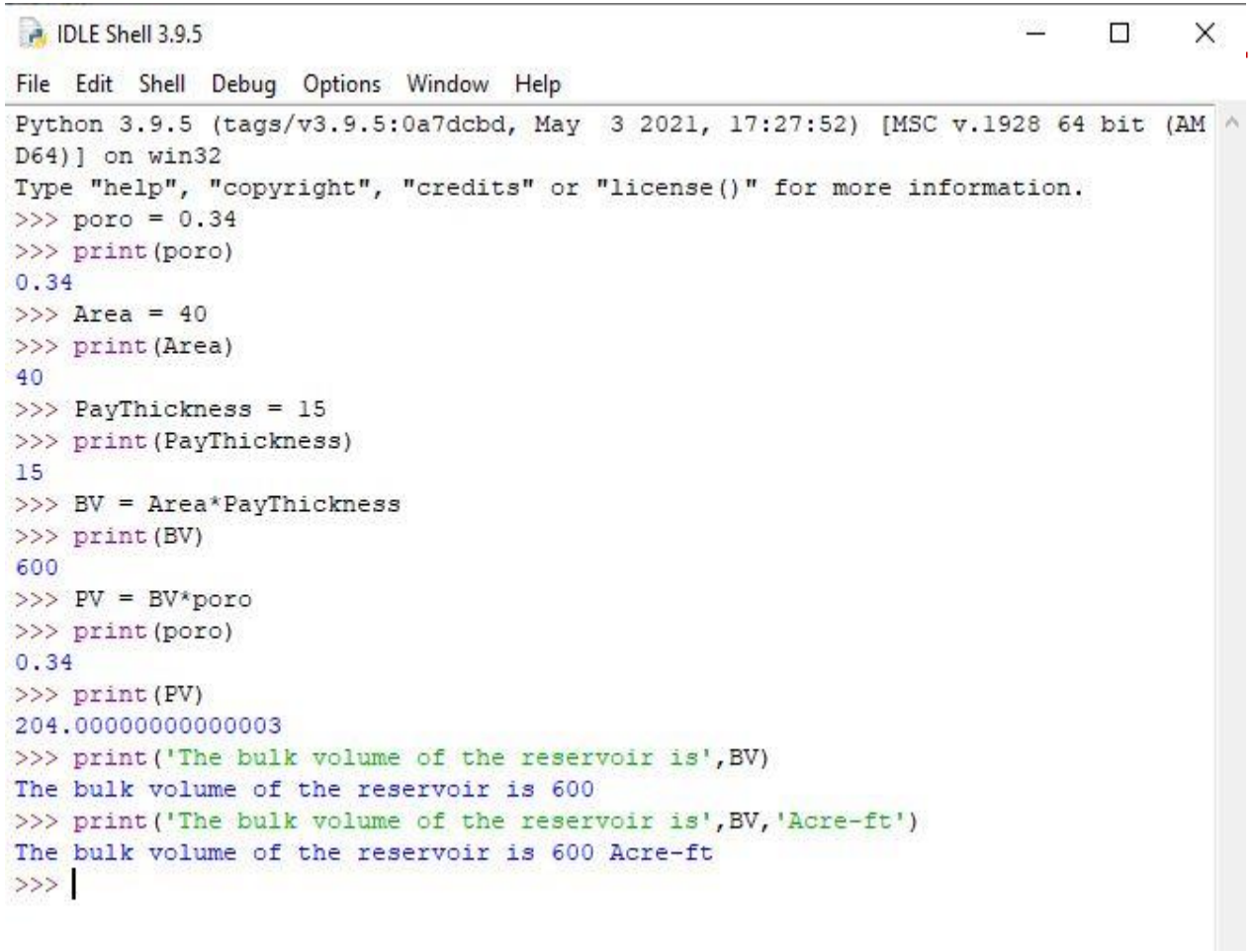
- Interactive
- From a file (script)

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.

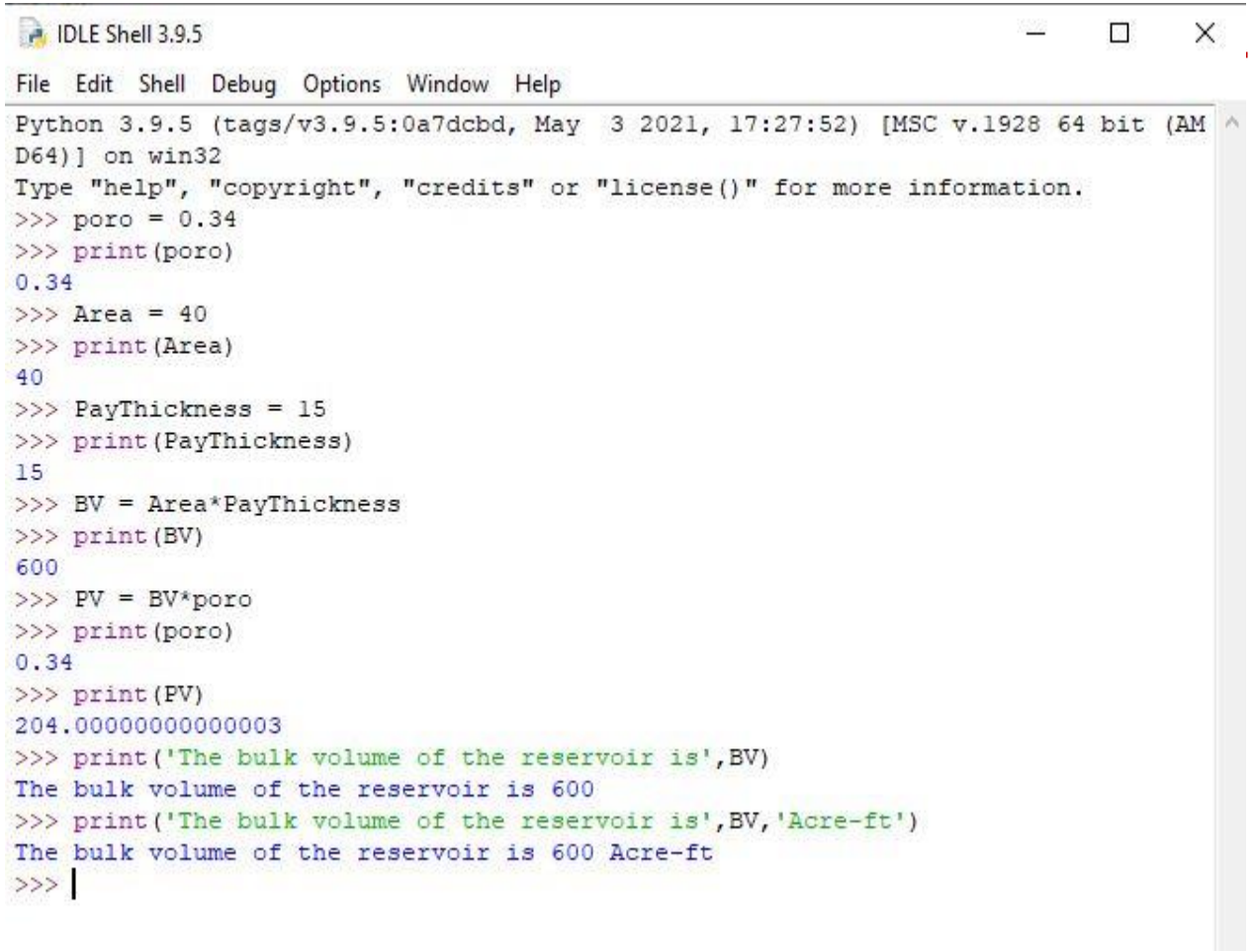


PREAMBLES

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.



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.

PREAMBLES

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.

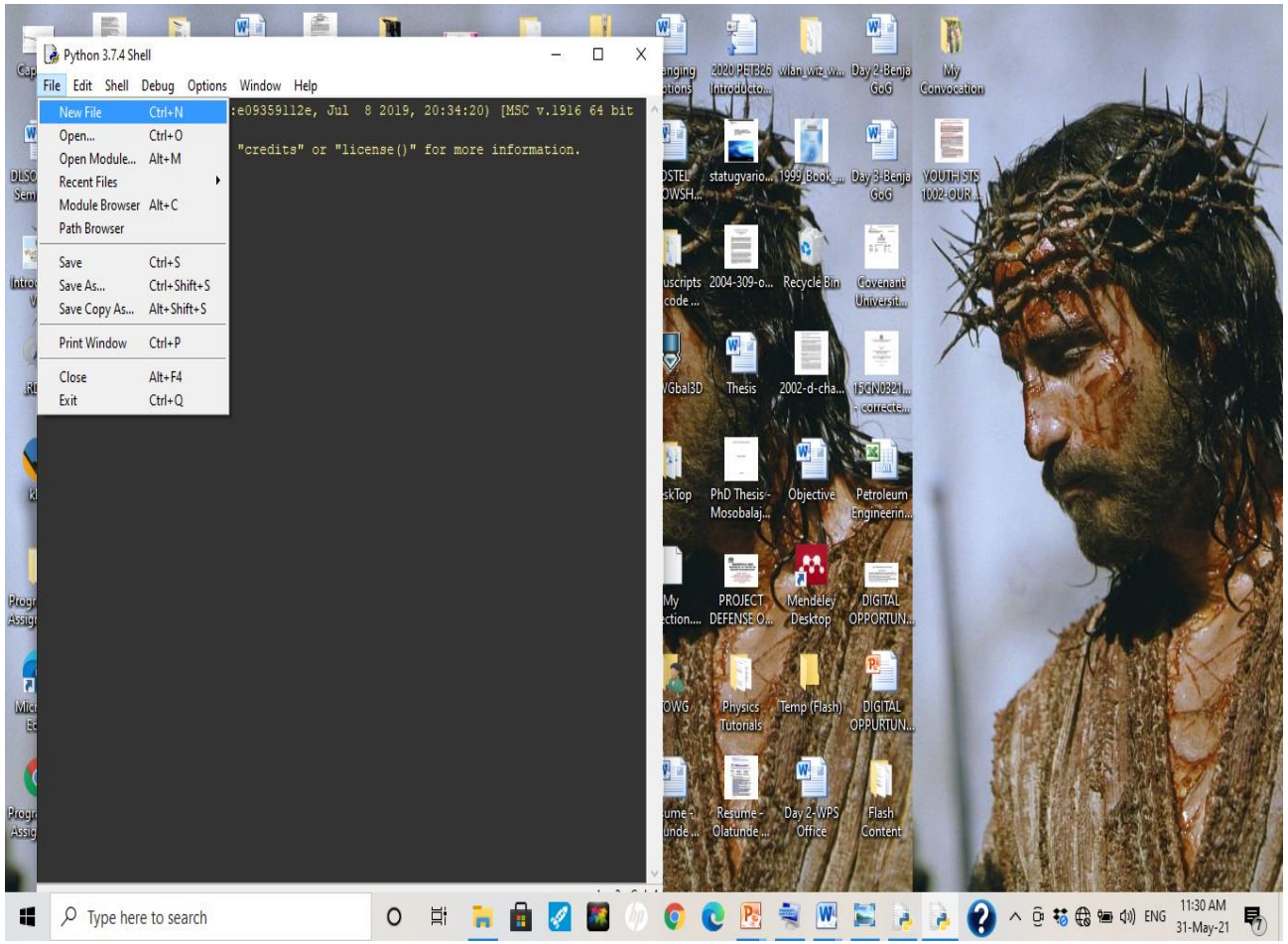
Good, Python has an in-built text editor for this purpose.

PREAMBLES

The Toolbox

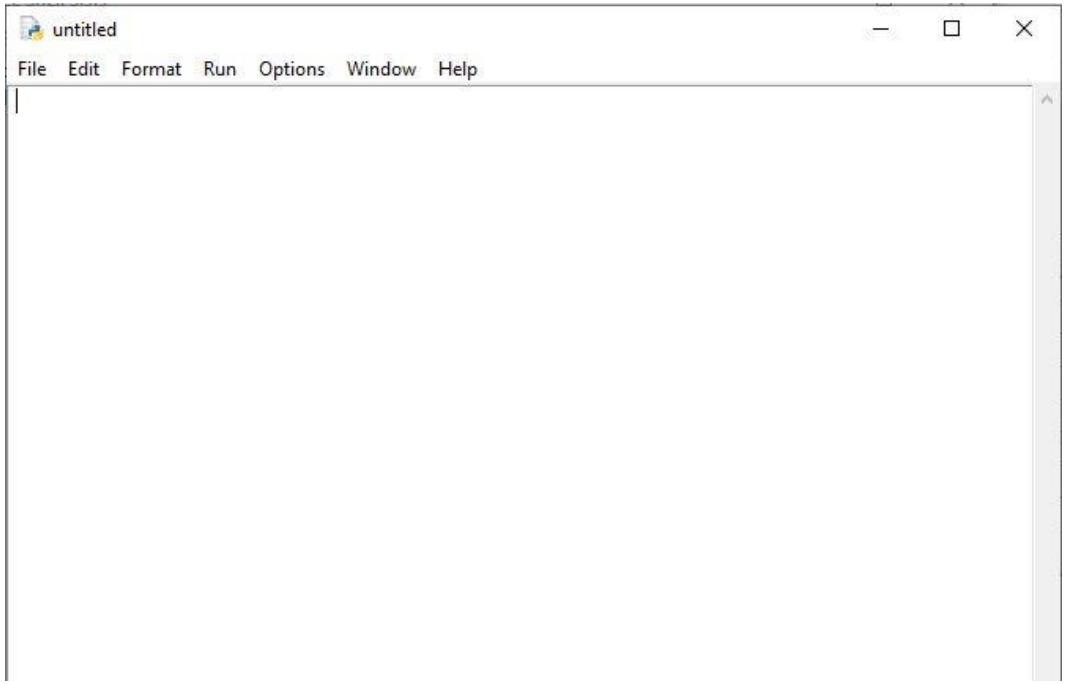
- Communicating with Python from a file

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

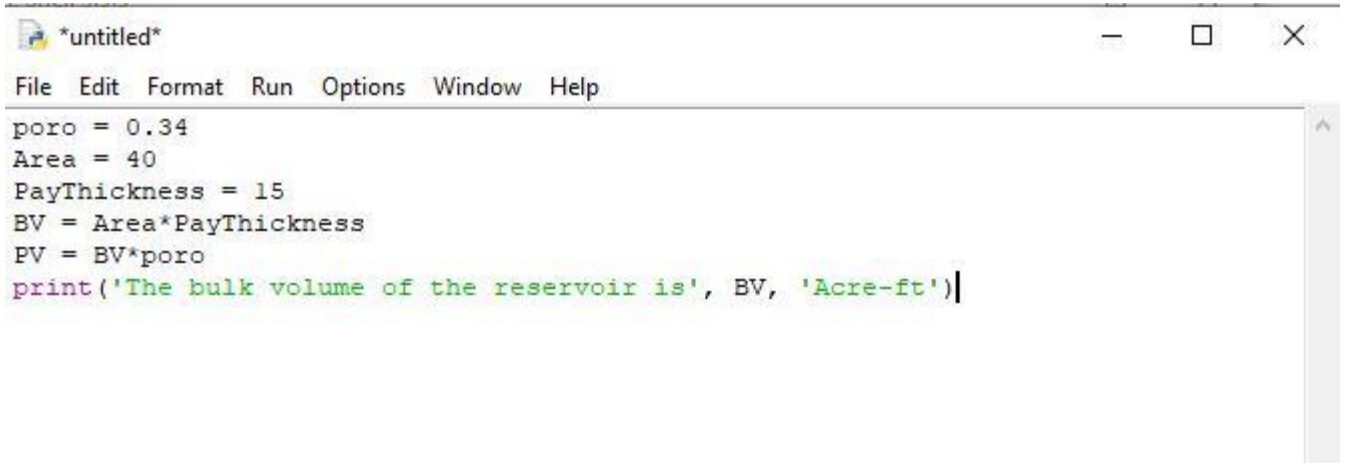


The Toolbox

- Communicating with Python from a file



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

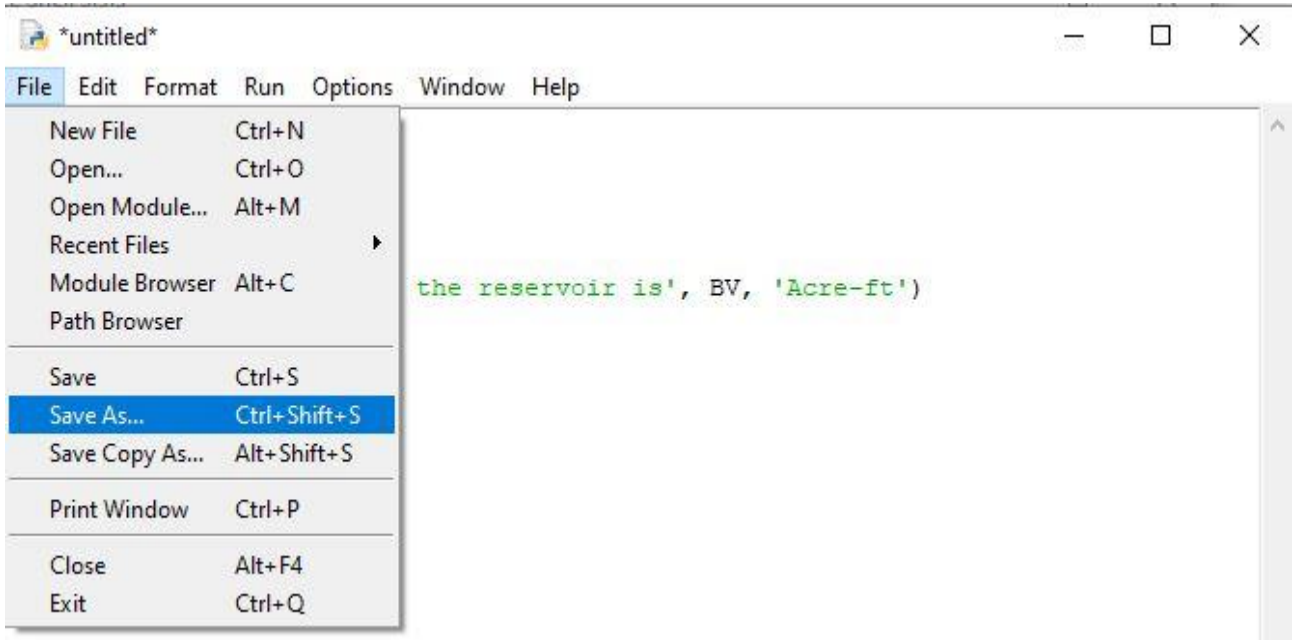


The Toolbox

- 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



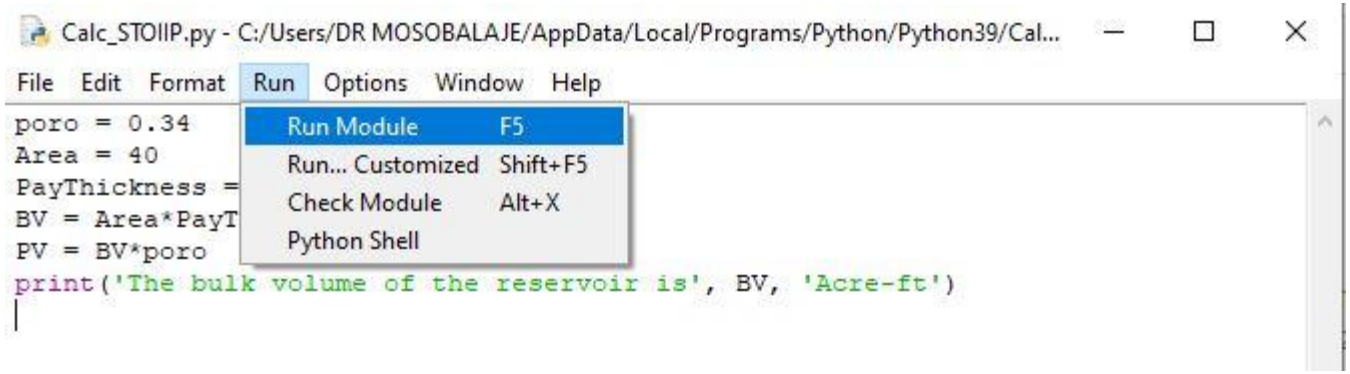
PREAMBLES

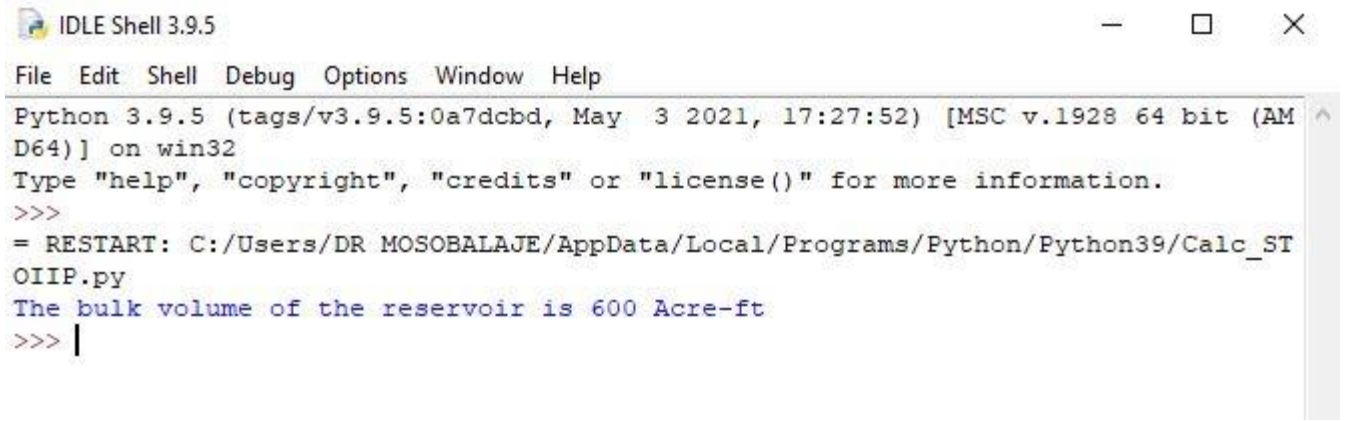
The Toolbox

- 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.

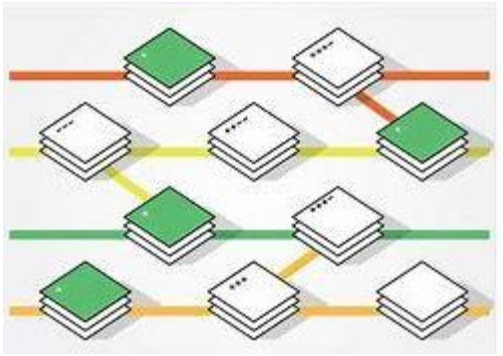




PREAMBLES

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.

PREAMBLES

The Toolbox

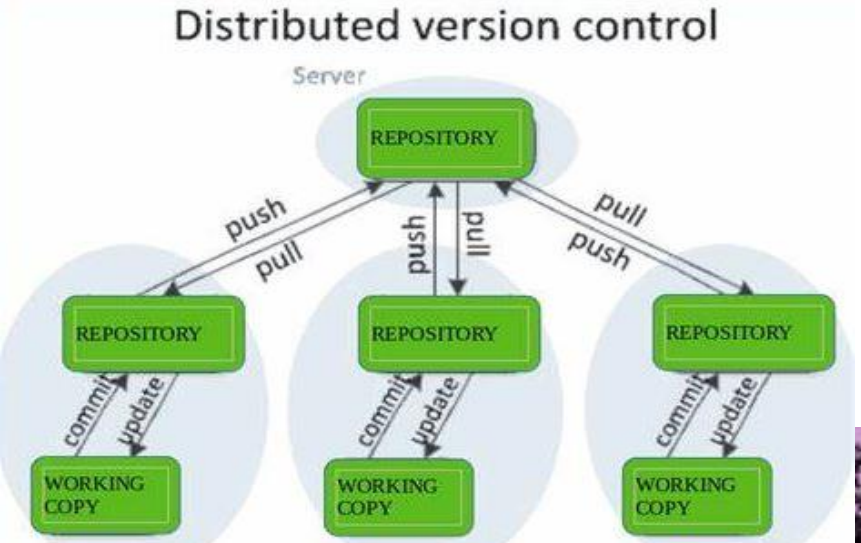
- Git and GitHub

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.

PREAMBLES

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)

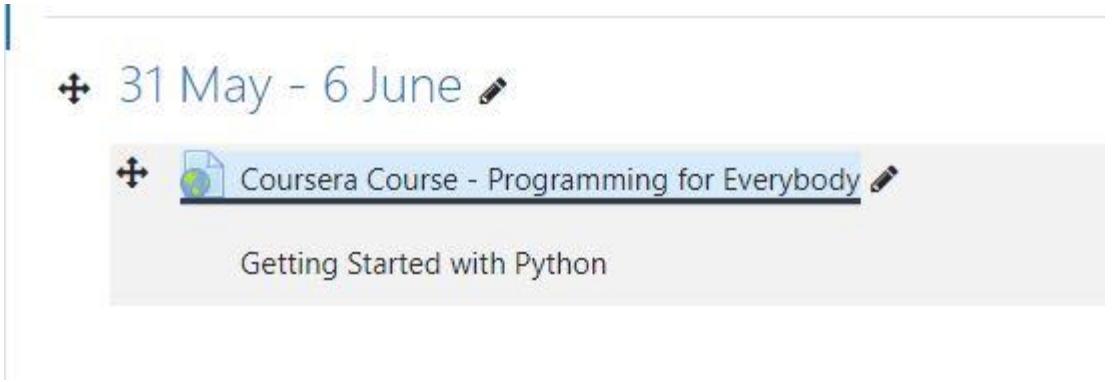
Offered by

UNIVERSITY OF MICHIGAN

PREAMBLES

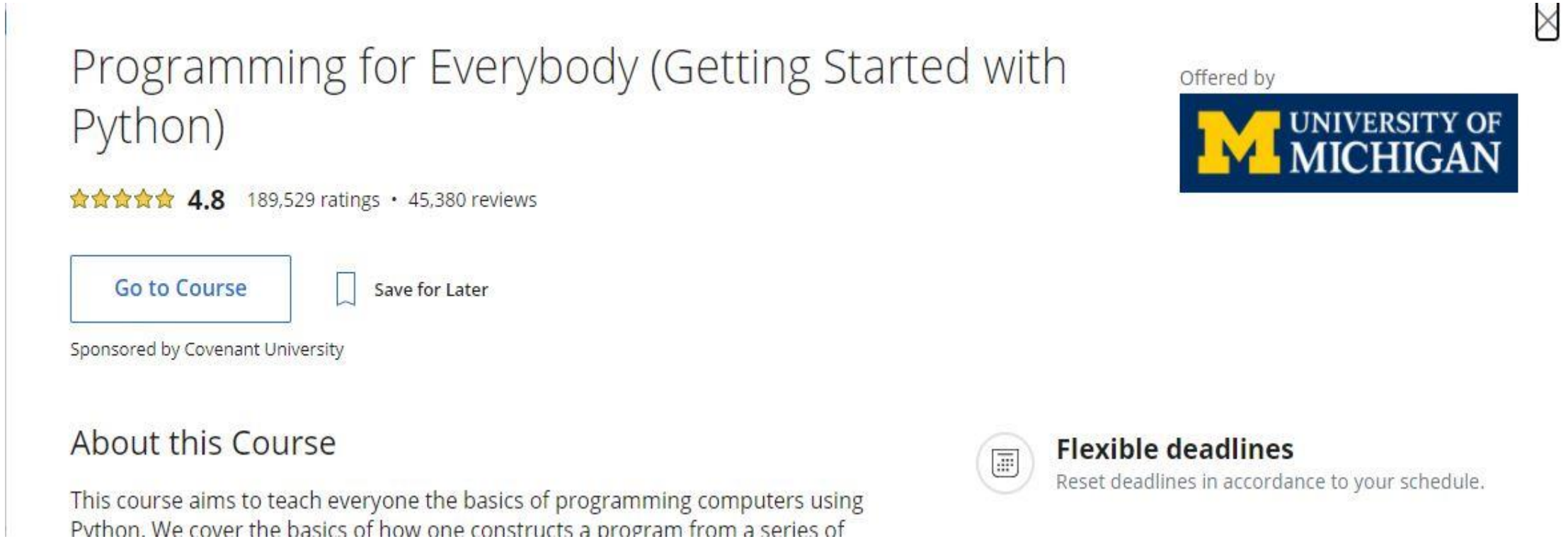
The Embedded Course

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.



PREAMBLES

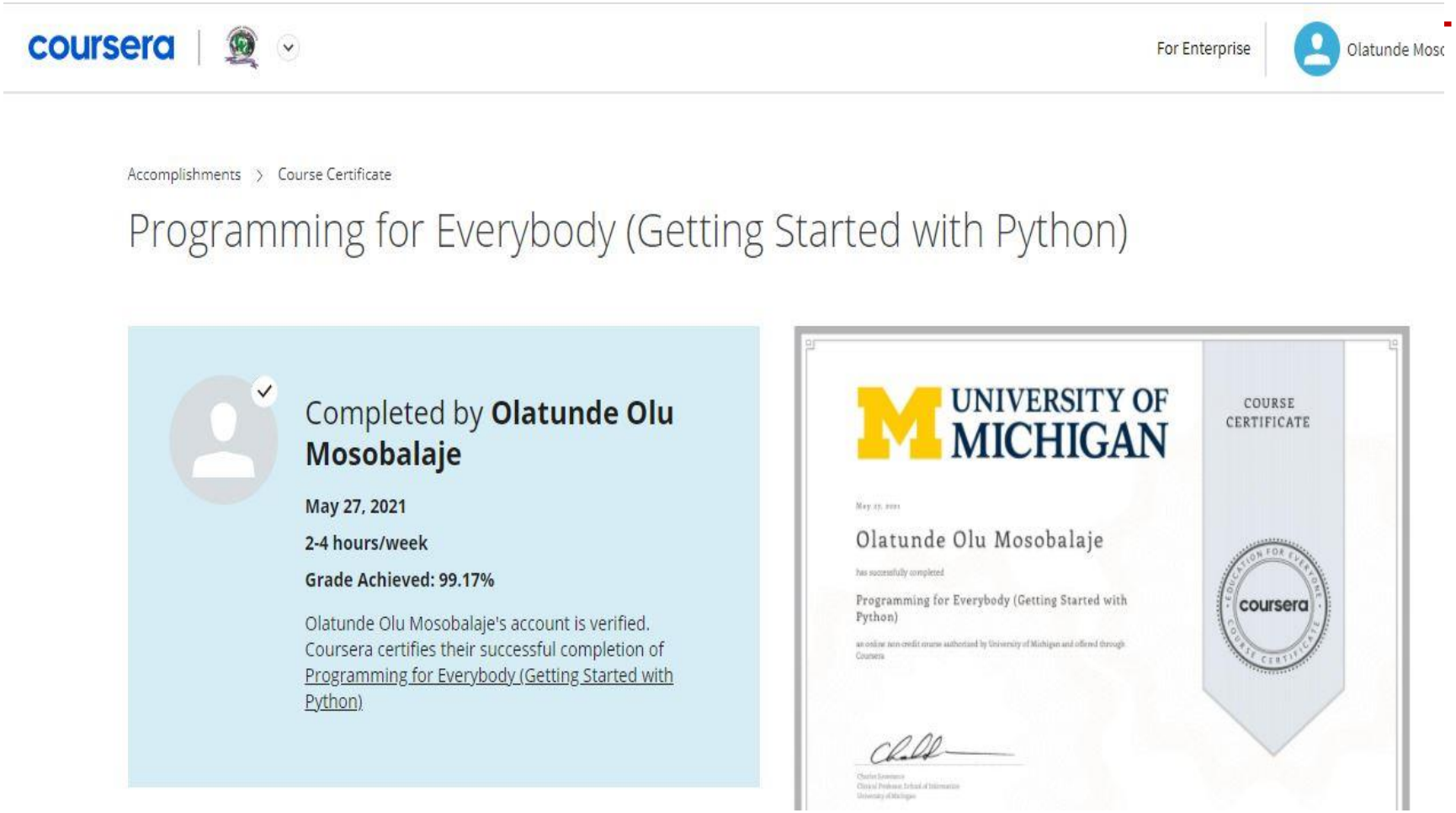
The Embedded Course



PREAMBLES

The Embedded Course

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



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_na
        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')
```

This script is available here

https://github.com/TTOWG/PET328_2021_Class/blob/main/demo_Input_Output.py

PREAMBLES

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

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')
```

PREAMBLES

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

File Edit Format Run Options Window Help

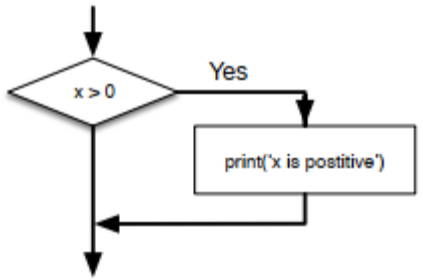
```
#...TTOWG!

# 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')
```

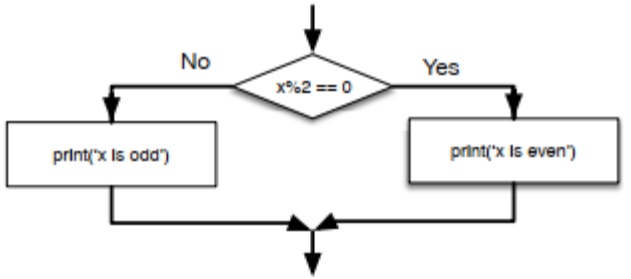
PREAMBLES

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

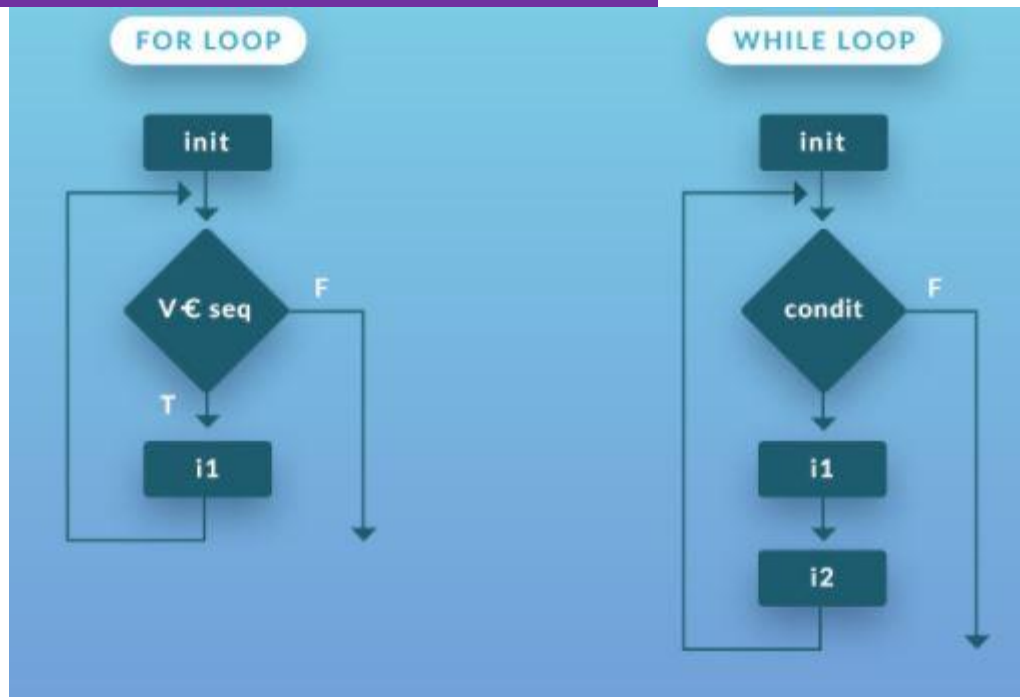
```
#...TTOWG!

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!!!')
```

PREAMBLES

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)
```

PREAMBLES

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...

File Edit Format Run Options Window Help

```
#...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 –

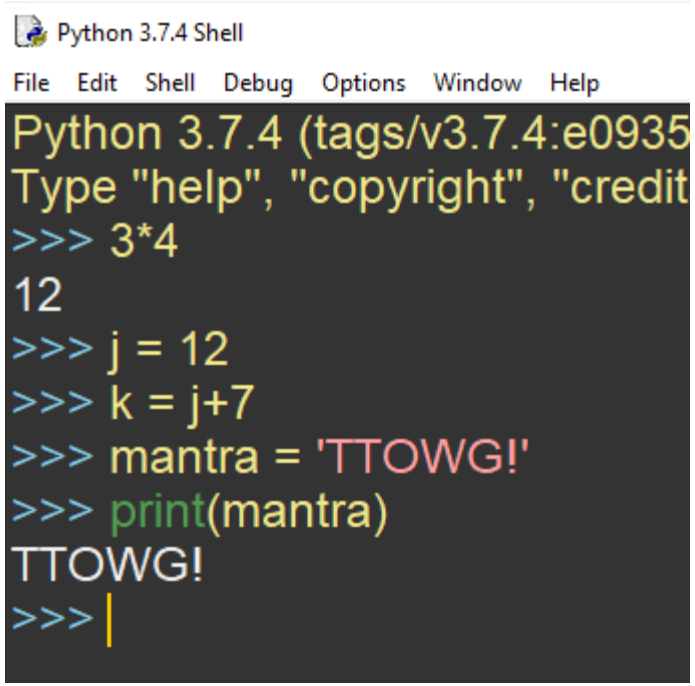
Getting Started with Python

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.



GETTING STARTED WITH PYTHON

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.

GETTING STARTED WITH PYTHON

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'>
```

GETTING STARTED WITH PYTHON

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.

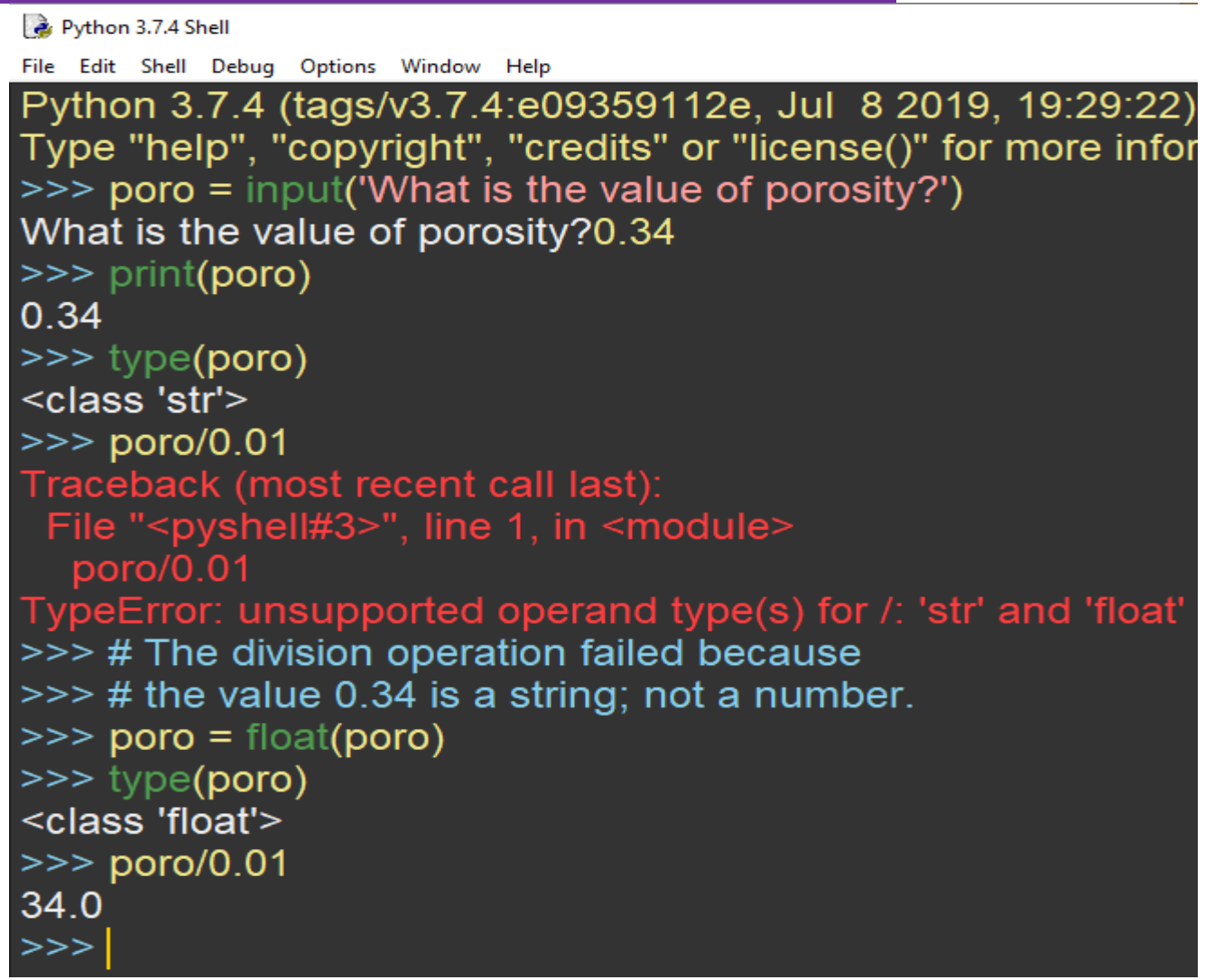
```
>>> type(2)
<class 'int'>
>>> type('TTOWG!')
<class 'str'>
>>> type(2.0)
<class 'float'>
>>> type('2')
<class 'str'>
```

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.



Basic Python Objects

Variables

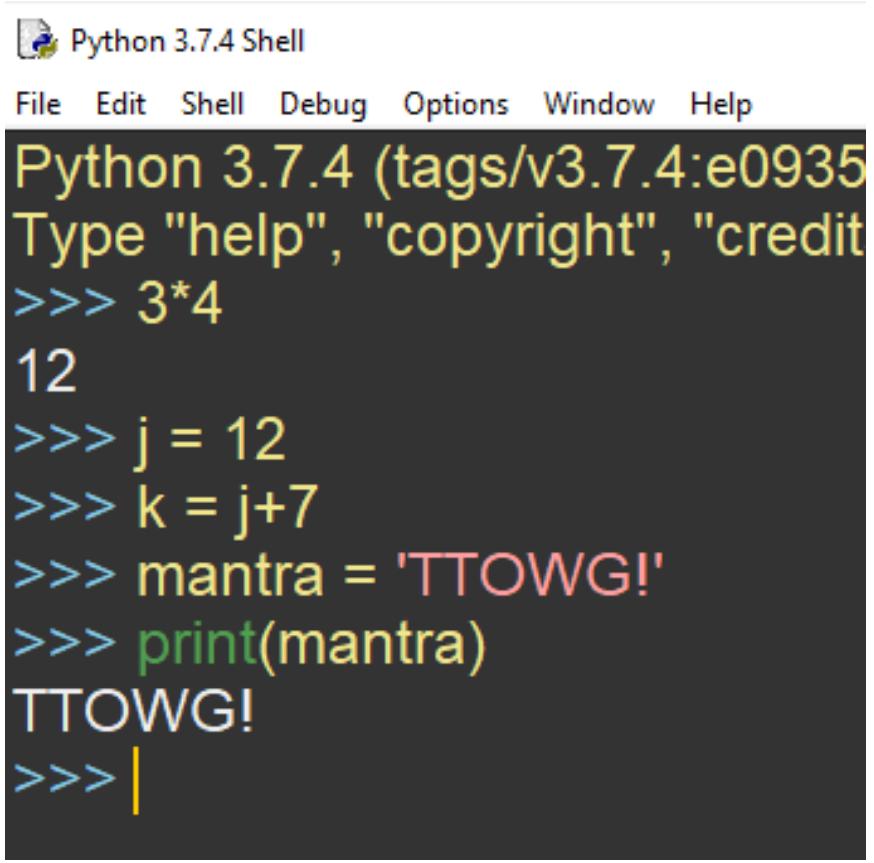
- A variable is a value stored in memory and referred to with a chosen name.
- In other words, values are assigned to variables.
- When the name of a variable is called, the value assigned therein answers.

GETTING STARTED WITH PYTHON

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.



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.

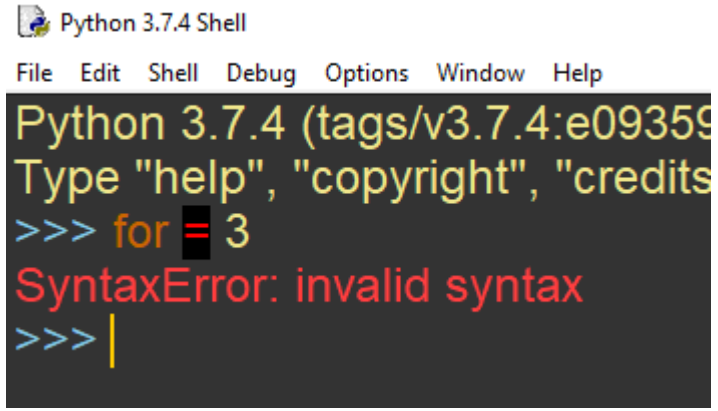
GETTING STARTED WITH PYTHON

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
```



GETTING STARTED WITH PYTHON

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
```

GETTING STARTED WITH PYTHON

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.

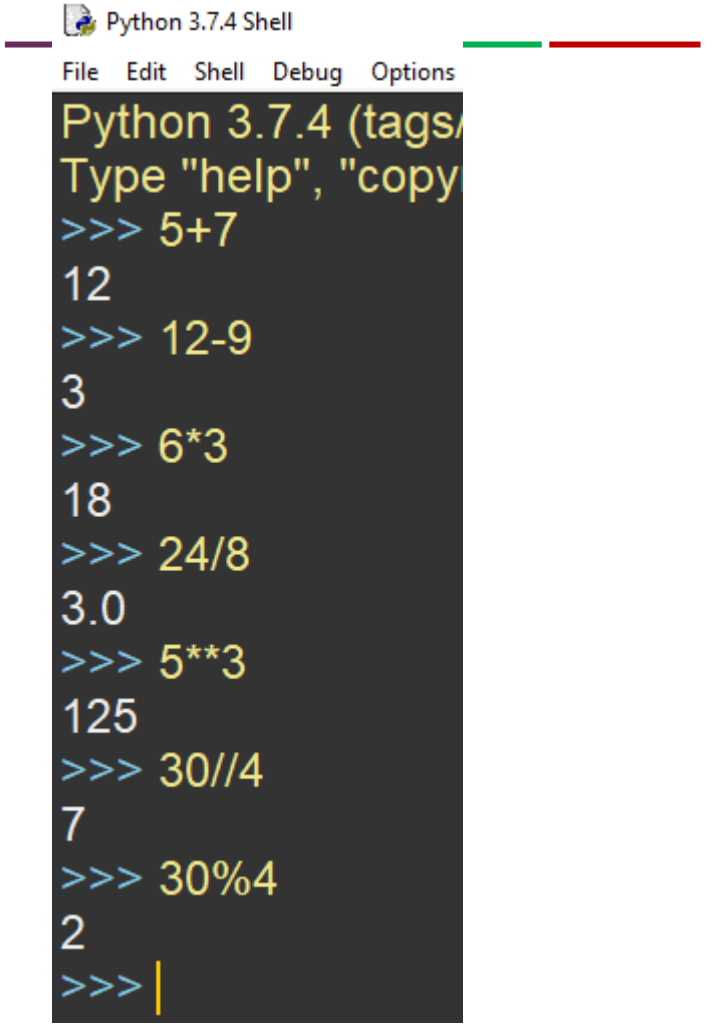
```
>>> poro = 0.18; area = 40; thickness = 15
>>> print(area)
40
>>>
```

GETTING STARTED WITH PYTHON

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.



Basic Python Objects

Order of Operations

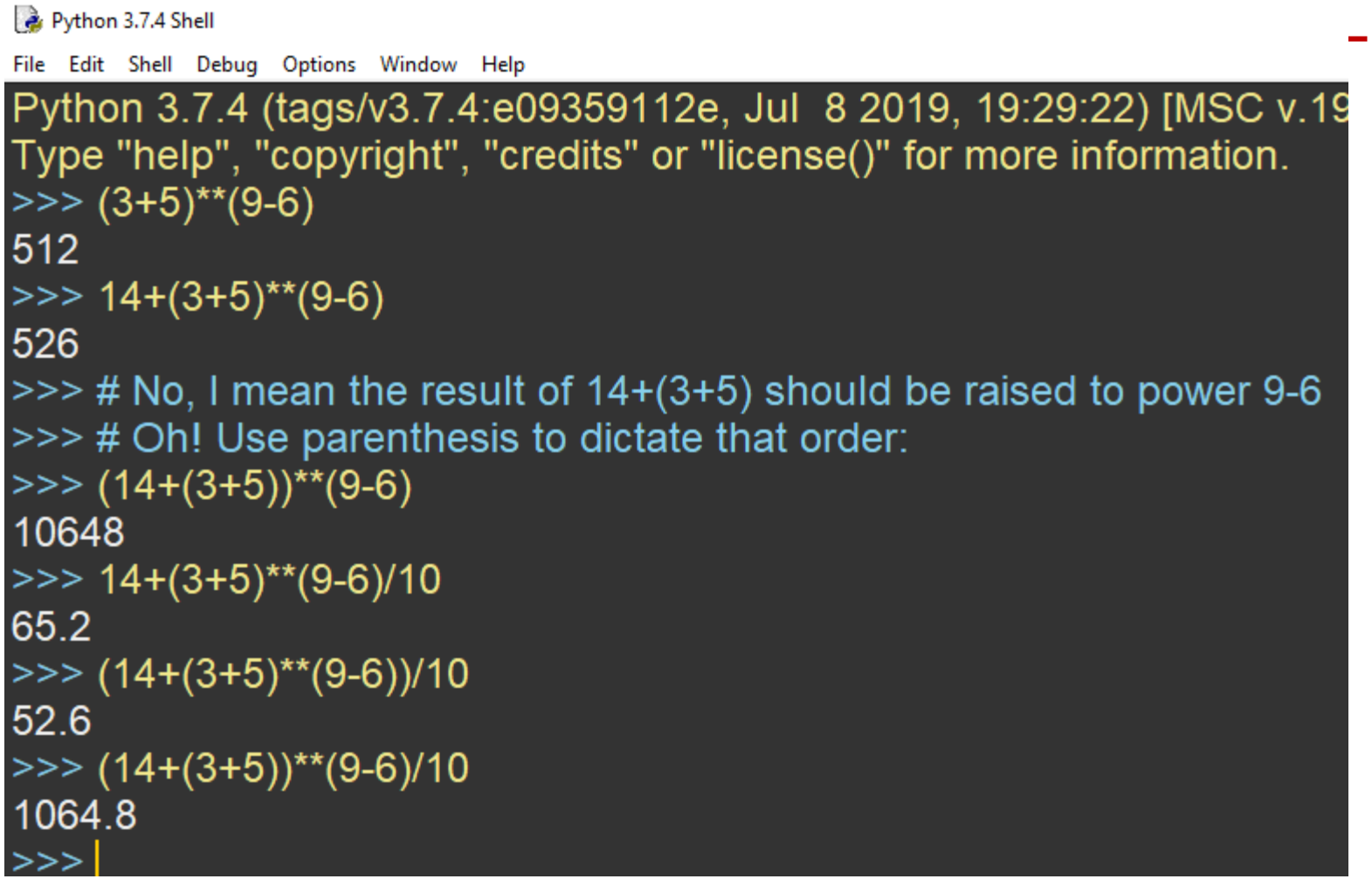
- When multiple operations are featured in a statement, Python executes them in the order specified by the accronym: PE-MD-AS (Parenthesis, Exponentiation, Multiplication, Division, and Subtraction).
- You can used parenthesis to dictate the order you desire.
- Multiplication and Division has equal precedence; 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.

GETTING STARTED WITH PYTHON

Basic Python Objects

Order of Operations

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



GETTING STARTED WITH PYTHON

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! '
>>>
```

GETTING STARTED WITH PYTHON

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
```

GETTING STARTED WITH PYTHON

Conditional Statements

Logical Operators

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

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

- 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.

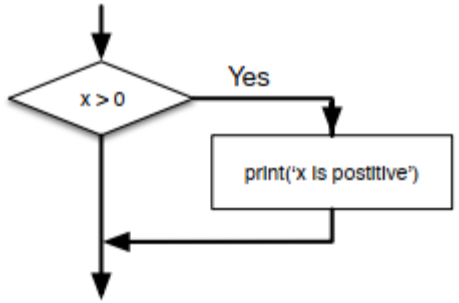
- Logical operators are also used to negate boolean expressions
  - *not* returns True for a false condition and vice-versa

GETTING STARTED WITH PYTHON

Conditional Statements

if… Statement (Conditional Execution)

- 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
```

Conditional Statements
if… Statement (Conditional Execution)

Petroleum engineering application

- Computing pseudo-critical gas properties using Sutton's correlation
  - Sutton developed a correlation for estimating for estimating P_{pc} and T_{pc} as functions of gas gravity.
  - Here is the first step in Sutton's procedure:
    - If the gas mixture contains <12 mol% of CO_2, < 3% of Nitrogen and no H_2S, then the parameter γ_h takes the same value as the given separator gas gravity (γ_g) or the given well-stream gravity (γ_w)
    - However, if gas mixture contains >12 mol% of CO_2, >3% of Nitrogen and any H_2S, then the parameter γ_h is determined thus:

$$\gamma_h = \frac{\gamma_w - 1.1767 y_{H_2S} - 1.5196 y_{CO_2} - 0.9672 y_{N_2} - 0.622 y_{H_2O}}{1 - y_{H_2S} - y_{CO_2} - y_{N_2} - y_{H_2O}}.$$

Conditional Statements
if… Statement (Conditional Execution)

Petroleum engineering application

- Computing pseudo-critical gas properties using Sutton's correlation
  - The first step in Sutton's can be executed with an if … statement.
  - Observe that the procedure implies that if any of the impurities in the gas exceeds the stated threshold value, then, the given gas gravity (γ_w) need to be corrected for the effects of the impurities, using the given equation.
  - However, the correction task should be neglected if none of the impurities exceeds its threshold value.

Conditional Statements
if… Statement (Conditional Execution)

Petroleum engineering application

- Computing pseudo-critical gas properties using Sutton's correlation
  - To execute this procedure, we simply construct a Boolean condition to test if any threshold is violated.
  - If the condition is evaluated as True, then a block of statement to perform the gs gravity correction is executed.
  - If the condition is evaluated to be False, there is no need for the correction, hence, the block of statement is skipped.

GETTING STARTED WITH PYTHON

Conditional Statements

if… Statement (Conditional Execution)

Petroleum engineering application

- Computing pseudo-critical gas properties using Sutton's correlation

```
if co2_comp > 0.12 or n2_comp > 0.03 or h2s_comp > 0:
    gas_gravity = (gas_gravity - (1.1767*h2s_comp) - \
                  (1.5196*co2_comp) - (0.9672*n2_comp) - \
                   (0.622*h2o_comp))/(1- h2s_comp - co2_comp - n2_comp - h2o_comp)
    print('The corrected gas gravity is', gas_gravity)
```

The full script for this computation is available here

Conditional Statements
if… Statement (Conditional Execution)

Assignment 2

Upgrade the *demo_gas_grav_corr.py* script (hosted on *TTOWG/ PET328_2021_Class* GitHub repository) to perform the entire Sutton's procedure. Save the upgraded script as *sutton_correlation.py*, commit and push it to your GitHub repository. Submit the URL to your copy of *PET328_2021_Class* repository. Furthermore, send a pull request to the original *TTOWG/ PET328_2021_Class* repository.

- The complete Sutton's algorithm is available here.

Conditional Statements
if… Statement (Conditional Execution)

Assignment 2

You may test run your script with the following data:

- **Inputs**
  - $y_{CO2} = 0.0164$
  - $y_{N2} = 0.0236$
  - $y_{H2S} = 0.1841$
  - Gas gravity = 0.6992

Outputs

- Corrected gas gravity = 0.5604
- Ppch = 682.3 psia.
- Tpch = 341.8 deg Rankine
- Ppc = 799.0 psia.
- Tpc = 403.3 deg Rankine

GETTING STARTED WITH PYTHON

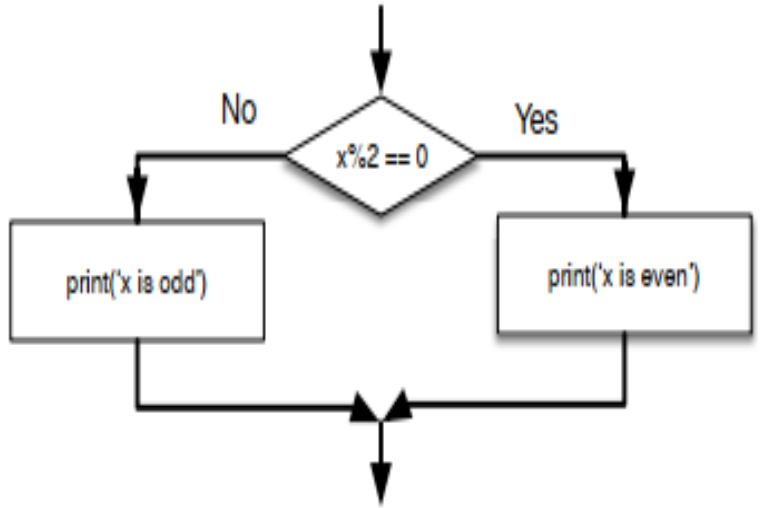
Conditional Statements

if…then…else Statement (Alternative Execution)

- The if…then…else structure is deployed when there are two alternative tasks and a condition that determines which of the two alternatives should be executed.

- Essentially, there will be a Boolean condition, and two blocks (branches) of statements.

- The first branch (after the condition) is to be executed if the condition evaluates to True while the second branch (after the keyword 'else') is executed if the condition evaluates to False.

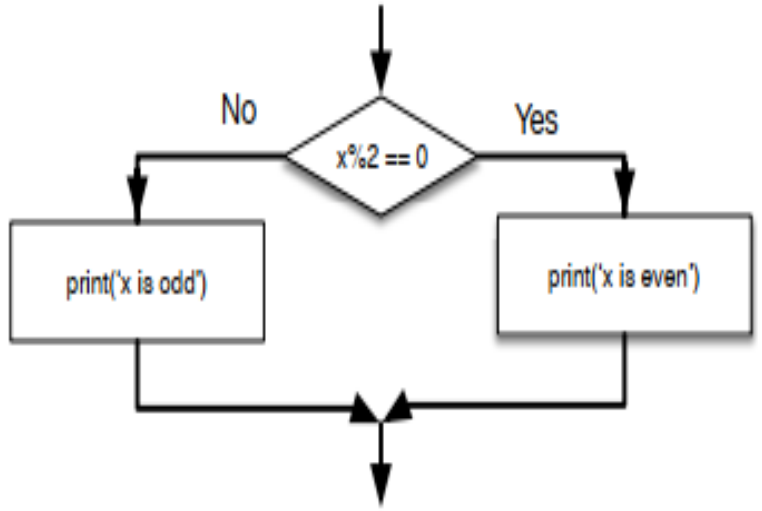


GETTING STARTED WITH PYTHON

Conditional Statements

if…then…else Statement (Alternative Execution)

- The condition(s) is written after the *if* keyword and ended with a colon i.e. (:)
- Then, the statement(s) to be performed if condition is True (i.e., Branch True) are written as an indented block in subsequent line(s).
- Thereafter, the keyword 'else' is written on the next line just after the Branch True. The 'else' keyword should be indented to the same level as the 'if' keyword.
- Finally, the statement(s) to be performed if condition is False (i.e., Branch False) are written as an indented block in subsequent line(s).



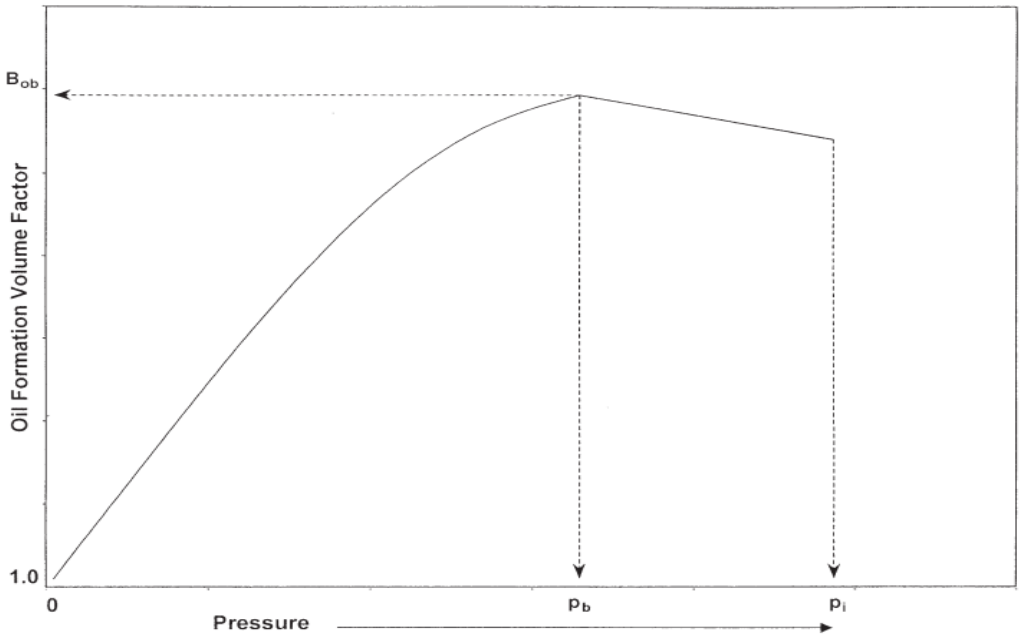

```
>>> perm = 90
>>> if perm < 50:
        print('Fair!')
else:
        print('Good!')


Good!
>>>
```

GETTING STARTED WITH PYTHON

Conditional Statements
if…then…else Statement (Alternative Execution)

Petroleum engineering application

- Computing oil formation volume factor, Bo.
  - The variation of oil formation volume factor, Bo, with pressure is divided into two pressure regimes, as shown.



Conditional Statements

if…then…else Statement (Alternative Execution)

Petroleum engineering application

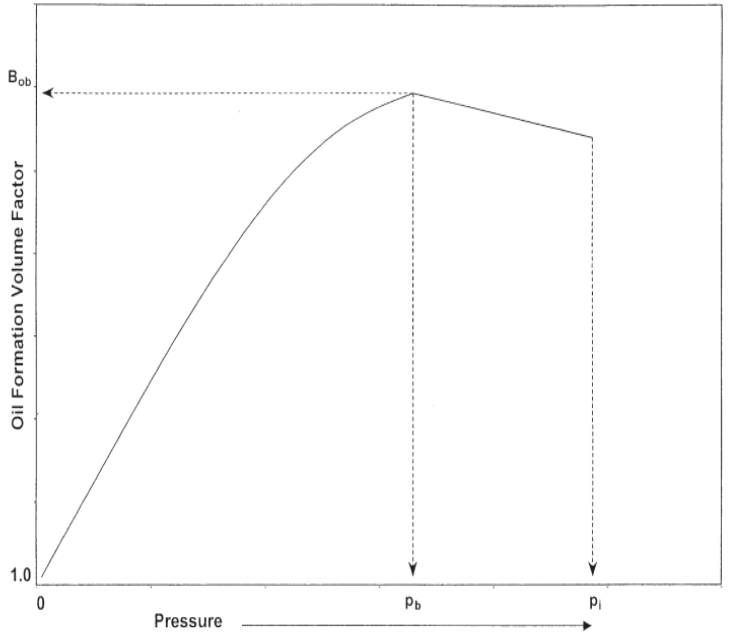
- Computing oil formation volume factor, Bo.

For pressures below or equal to bubble point, Standing's correlation for calculating Bo is herein presented:

$$B_o = 0.9759 + 0.00012F^{1.2} \text{ ----- } -2.35$$

$$Where\ F = R_s \left(\frac{\gamma_g}{\gamma_o}\right)^{0.5} + 1.25T_F \text{ -- } -2.36$$

Note: T_F is temperature in degree Fahrenheit.



GETTING STARTED WITH PYTHON

Conditional Statements if…then…else Statement (Alternative Execution)

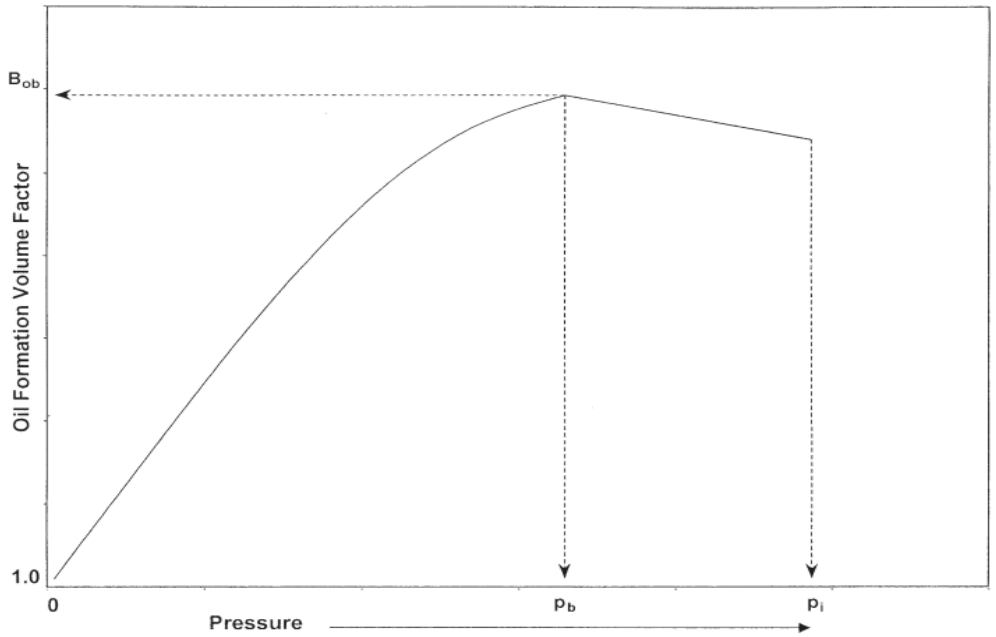
Petroleum engineering application

- Computing oil formation volume factor, Bo.

For pressure above bubble point, the analytical equation for computing Bo is given as:

$$B_o = B_{ob} e^{[c_o(P_b - P)]} \text{ ---} -2.37$$

Bob is the Bo at bubble point and can be computed using Equations 2.35 and 2.36



Conditional Statements

if…then…else Statement (Alternative Execution)

Petroleum engineering application

- Computing oil formation volume factor, Bo.
  - To execute this procedure, we simply construct a Boolean condition to test if the current reservoir pressure, p, is greater than the bubble-point pressure of the reservoir.
  - If the condition is evaluated as True, then a block of statement to implement Equation 2.37 is executed.
  - Else, if the condition is evaluated to be False, then a block of statement to implement Equation 2.35 is executed.
  - Note that Equation 2.36 need to be implemented for either of the alternatives, hence, the line to execute it is written before the if…then…else statement.

GETTING STARTED WITH PYTHON

Conditional Statements

if…then…else Statement (Alternative Execution)

Petroleum engineering application

- Computing oil formation volume factor, Bo.

```
# calculating F parameter
F = (rs*((gas_gravity/oil_gravity)**0.5))+(1.25*tf)

# the if-then-else statement

if p > pb:
    bob = 0.9759+(0.00012*(F**1.2))
    bo = bob*(math.exp(co*(pb-p)))
else:
    bo = 0.9759+(0.00012*(F**1.2))
```

The full script for this computation is available here.

Conditional Statements
Reading Assignment

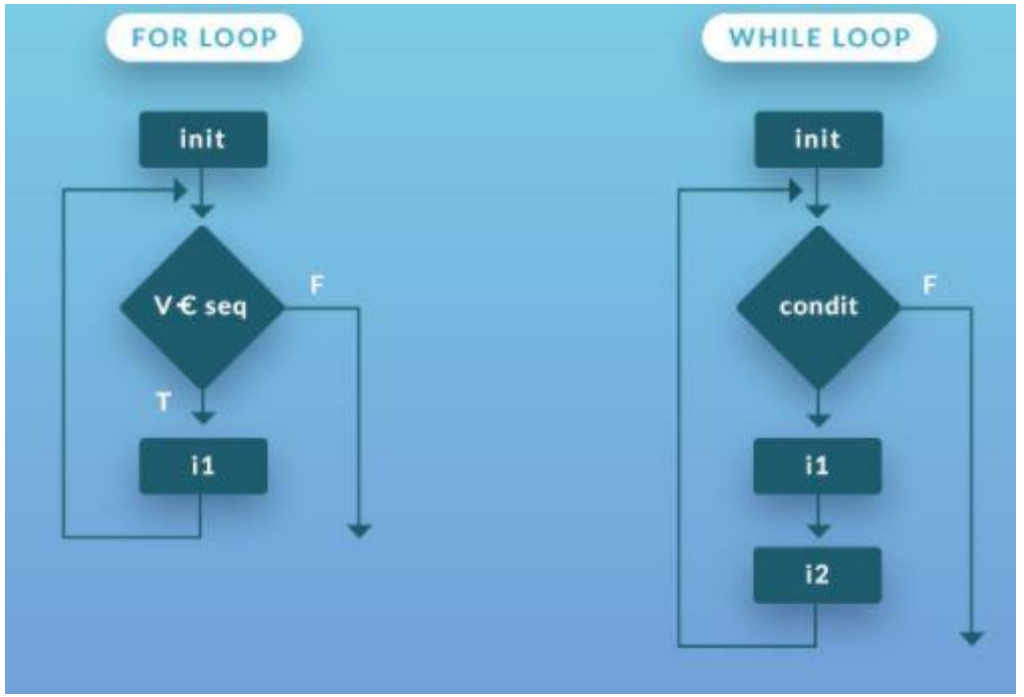
- Read on Chained Conditionals (if…elif… statements) and Nested Conditionals, in the recommended textbook for this course (pages 34 – 36)

Recommended Textbook: *Python for Everybody: Exploring Data using Python 3*, by Charles Severance.

GETTING STARTED WITH PYTHON

Repeated Execution

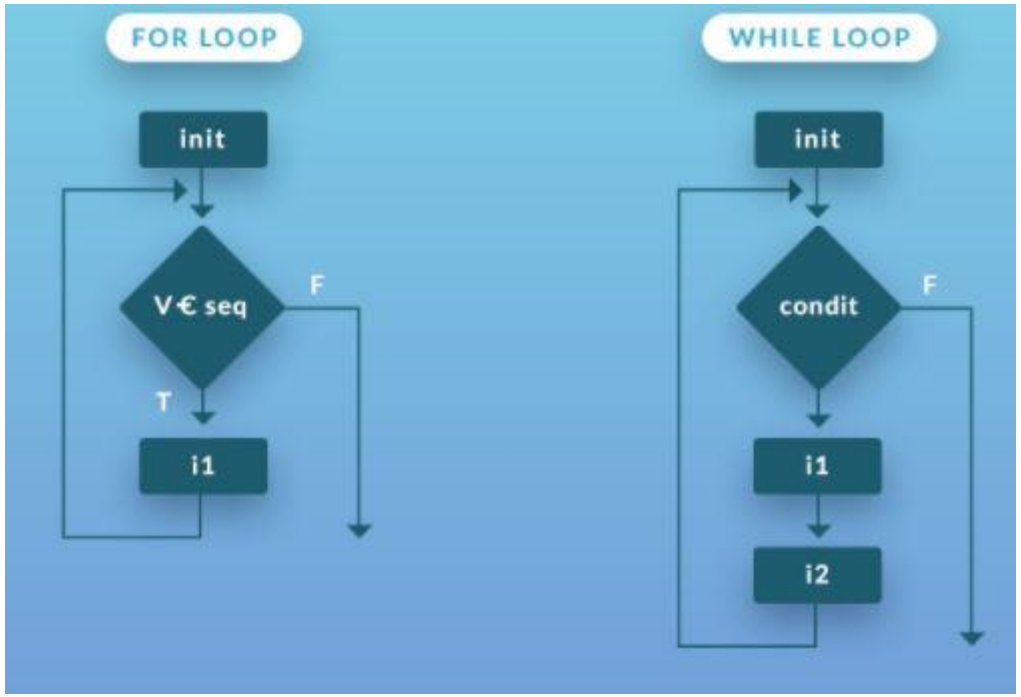
- One major reason for writing computer programs is to automate repetitive workflows.
- When a given task is to be performed repeatedly for each member of a set of entities, that is a repetitive workflow.
- In implementing such workflows, moving from one entity to the next entity is called looping or iterating.



GETTING STARTED WITH PYTHON

Repeated Execution

- Sometimes, the list of entities in the set is known explicitly and presented to the computer – that is a definite loop.
- At other times, the list is not known explicitly, rather the computer is asked to perform the task repeatedly until a given condition becomes False – that is an indefinite loop.
- Definite loops are implemented with 'for' loops and indefinite loops are implemented with 'while' loops.



GETTING STARTED WITH PYTHON

Repeated Execution
for… loops

- In practice, 'for' loops are used in counting the number of items/elements in a list or in summing up (aggregating) the results of a computation for every item/element in a list.
- In such cases, a variable to hold the count or sum is normally, created and set to a dummy initial value before the 'for' loop.
- The first line (header) of a 'for' loop begins with the keyword 'for', followed by an iterator variable, followed by the keyword 'in' and lastly, the list of items on which the task is to be performed.
- Statement(s) to implement the task are written as an indented block on subsequent lines after the header.

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


This is Block 1
This is Block 2
This is Block 3
This is Block 4
This is Block 5
>>>
```

GETTING STARTED WITH PYTHON

Repeated Execution
for… loops

Iterator variable

- The iterator variable is a sort of temporary variable that represent the item/element of the set being treated in the current loop cycle.
- So, the iterator variable changes value as the program loops through the list of items.

- Sometimes, the in-built function *range* is used to generate the list of items.

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


This is Block 1
This is Block 2
This is Block 3
This is Block 4
This is Block 5
>>>
```

```
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list(range(1,5))
[1, 2, 3, 4]
>>> list(range(1,6))
[1, 2, 3, 4, 5]
>>>
```

```
>>> for point in range(1,6):
        print('This is Point', point)


This is Point 1
This is Point 2
This is Point 3
This is Point 4
This is Point 5
```

Repeated Execution
for… loops

Petroleum engineering application

- Reservoir discretization
  - Input parameters required to solve reservoir engineering models are essentially rock and fluid properties.
  - These rock and fluid properties are known to vary across the reservoir – heterogeneity.
  - The challenge: which value of a property is to be used in solving the model for a given reservoir???
  - Average??? Nay!
  - Locally-acceptable values??? Yes!

GETTING STARTED WITH PYTHON

Repeated Execution

for… loops

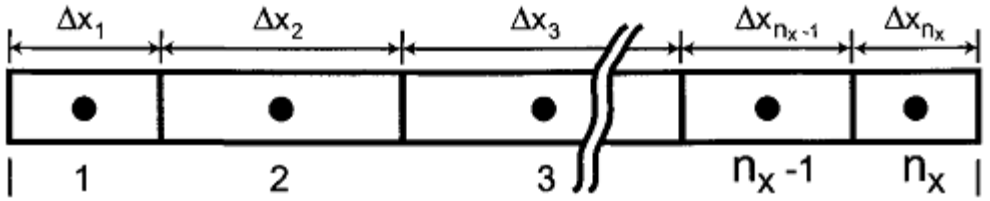
Petroleum engineering application

Reservoir discretization

- Discretization is the means by which locally-acceptable values of reservoir rock and fluid properties are honored in reservoir modelling.

- Loosely speaking, reservoir discretization is the division of the reservoirs into grid-blocks whose properties, dimensions and locations are well defined and uniform.

- Upon discretizing the reservoir into blocks; the flow model would then need be written and solved, repeatedly, for each block.



Repeated Execution
for… loops
Petroleum engineering application

- Reservoir discretization
- Blocks Ordering (Numbering) Schemes –
  - a way to identify each block in 1D, 2D or 3D discretized model.
  - Two types of ordering:
    - Engineering ordering.
    - Natural ordering

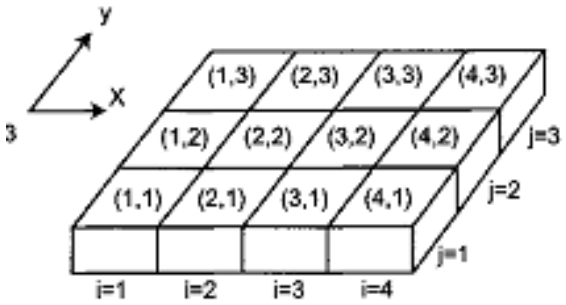
GETTING STARTED WITH PYTHON

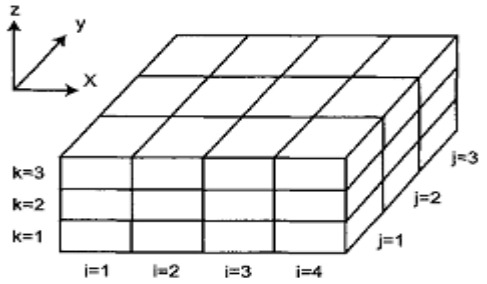
Repeated Execution

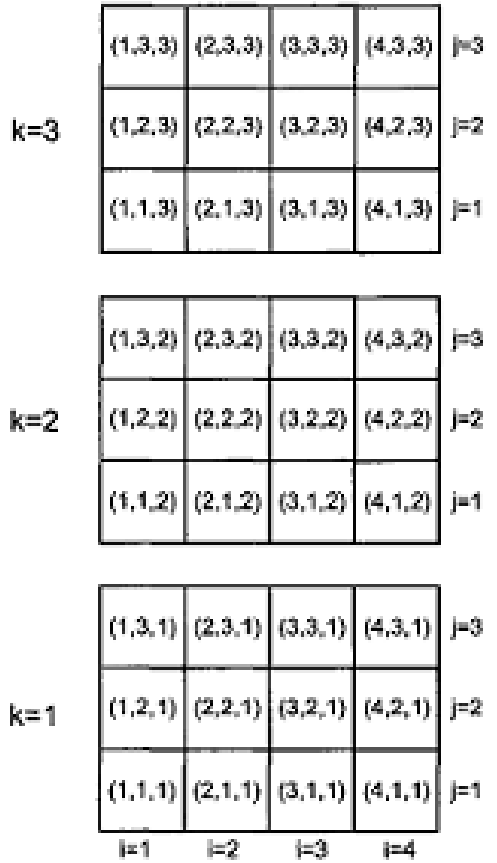
for… loops

Petroleum engineering application

- Reservoir discretization
- Engineering Ordering –
  - uses i, j, k, notations to order blocks in the x, y, z directions respectively.
    - i – counts columns along a certain row;
    - j – counts rows along a certain column;
    - k refers to layers.



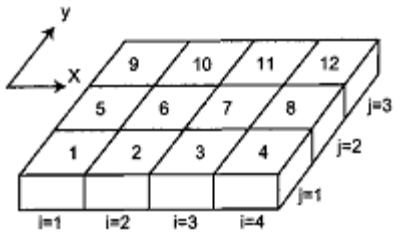


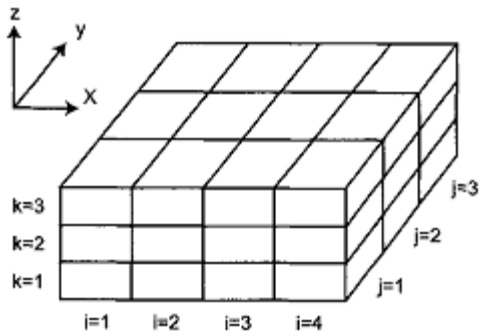


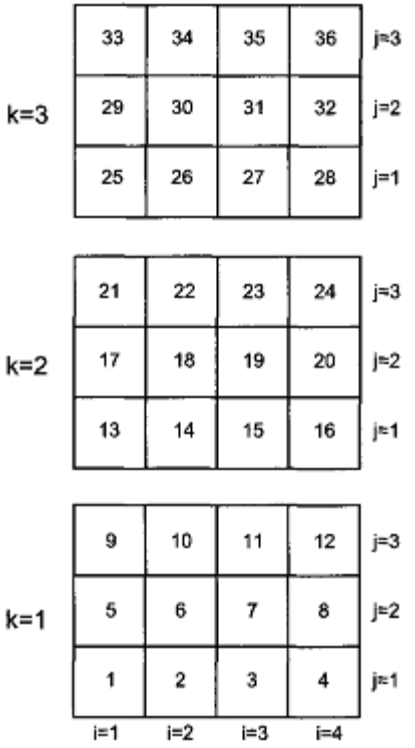
GETTING STARTED WITH PYTHON

Repeated Execution
for… loops
Petroleum engineering application

- Reservoir discretization
- Natural Ordering –
  - uses natural counting scheme.
  - Columns are counted fastest, followed by rows and then layers.







(c) Natural ordering of blocks.

Repeated Execution
for… loops

Petroleum engineering application

- Reservoir discretization
- The engineering ordering fits perfectly well into the 'for' loop scheme.
  - Looping through columns in a given row would be implemented by a column 'for' loop whose iterator variable would be *i* (the column counter).
  - Looping through rows in a given layer would be implemented by a row 'for' loop whose iterator variable would be *j* (the row counter).
  - Looping through layers would be implemented by a layer 'for' loop whose iterator variable would be *k* (the layer counter).

Repeated Execution
for… loops

Petroleum engineering application

- Reservoir discretization
- In a one-dimensional discretized model (i.e. a model discretized in only one direction), only one 'for' loop is needed; it may be column, row or layer loop.
- In a multi-dimensional discretized model (i.e. a model discretized in multiple directions), multiple nested 'for' loops are needed; with the column loop cycling faster than the row loop and the row loop cycling faster than the layer loop.
- In nested loop, the innermost loop cycles fastest while the outermost loop cycles the slowest.

Repeated Execution
for… loops

Petroleum engineering application

Reservoir discretization

- While the engineering ordering is used in computations, communicating outputs of such computations is better done with natural ordering.
- The following equation can be used to obtain the natural ordering index of a block from its engineering ordering indices (i, j, k)

$$n_{order} = [(k-1)n_x.n_y] + [(j-1)n_x] + i$$

  - n_{order} = natural order index of the block
  - i is the column index of the block
  - j is the row index of the block
  - k is the layer index of the block
  - n_x is the number of blocks in x-direction (i.e. number of columns)
  - n_y is the number of blocks in y-direction
- A constant value of 1 is used for any of the i, j, k indices not relevant to a case at hand.

GETTING STARTED WITH PYTHON

Repeated Execution

for… loops

Petroleum engineering application

- Computing STOIIP for a discretized reservoir model
- The amount of stock tank oil in-place in a given reservoir is given thus:

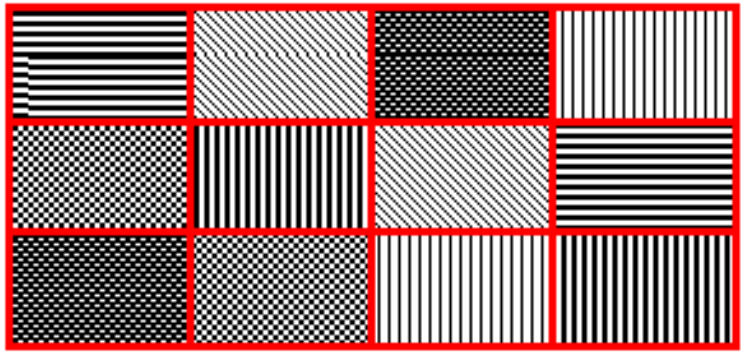
$$STOIIP = \frac{7758Ah\phi(1 - S_{wi})}{B_{oi}}$$

- Typically, values of porosity (ϕ)and initial water saturation varies across the reservoir; hence, the given reservoir is discretized and the STOIIP equation is implemented for each block of the discretized model. The STOIIP for each is block aggregated in a running total to yield the reservoir STOIIP ultimately.

GETTING STARTED WITH PYTHON

Repeated Execution
for… loops
Petroleum engineering application

- Computing STOIIP for a discretized reservoir model
- Here is a simple (trivial) 2D example of the grid of such discretized reservoir models.



Legend	poro	swi
	0.1	0.23
	0.25	0.29
	0.29	0.31
	0.33	0.37
	0.23	0.20
	0.27	0.28

GETTING STARTED WITH PYTHON

Repeated Execution
for… loops

Petroleum engineering application

- Computing STOIIP for a discretized reservoir model

```
# the 'for' loop

for j in range(1,ny+1):
    for i in range(1,nx+1):
        block_n_order = (nx*(j-1))+i
        poro = float(input('What is the value of porosity for Block {0}?'.format(block_n_order)))
        sw = float(input('What is the value of water saturation for Block {0}?'.format(block_n_order)))
        block_stoiip = (7758*area*h*poro*(1-sw))/boi
        total_stoiip = total_stoiip + block_stoiip
        print('The amount of oil in Block {0} is {1:.2f} STB'.format(block_n_order, block_stoiip))
```

The full script for this computation is available here.

Disclaimer: the implementation of this task as presented in the script is only for pedagogical purposes; in reality, a more efficient implementation would be done.

GETTING STARTED WITH PYTHON

Repeated Execution
for… loops

Assignment 3

An undersaturated oil reservoir material balance simulator computes the cumulative oil produced from a block, in a discretized model, thus:

$$N_p = \frac{NB_{oi}c_e(P_i - P_{now})}{B_o} \text{ --------} -P5$$

- N_p is the cumulative oil produced from a given block.
- B_{oi} is the initial oil formation volume factor (assumed constant for all blocks)
- c_e is the effective compressibility (assumed constant for all blocks)
- P_i is the initial reservoir pressure (assumed constant for all blocks)
- P_{now} is the current reservoir pressure (varies across blocks depending on proximity to producer well)
- B_o is the current value of the oil formation volume factor (depends on current pressure in a block)

GETTING STARTED WITH PYTHON

Repeated Execution
for… loops

Assignment 3

- Below is the expression to calculate the B_o value corresponding to a given current pressure.

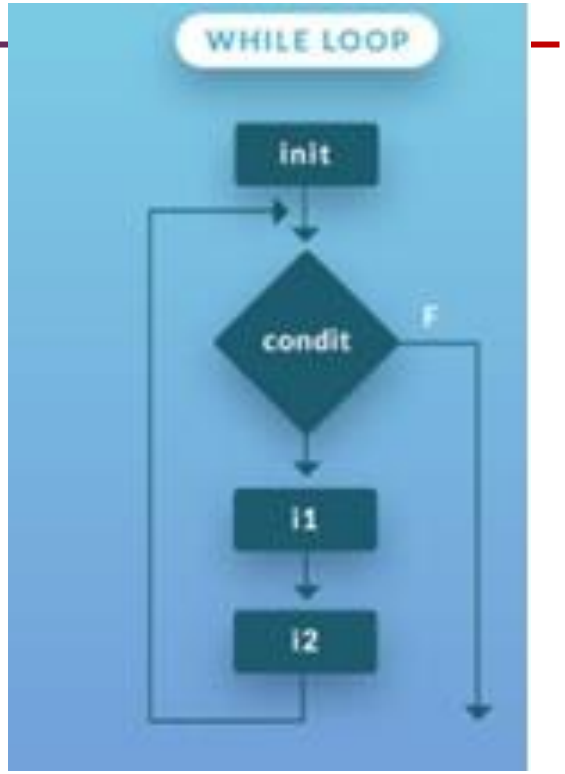
$$B_o = B_{ob}[1 - c_o(P_{now} - P_b)] \text{ ---------- } -P6$$

- Given a set of parameter (N_p, B_{oi}, B_{ob}, c_e, c_o and P_i) values and a grid of current pressure values, write a Python script to implement Equations P5 and P6 for each block. Also include statements to sum up and present the total cumulative oil produced from the entire reservoir.

- Save the script as *mat_bal.py*, commit and push it to your GitHub repository. Submit the URL to your copy of *PET328_2021_Class* repository. Furthermore, send a pull request to the original *TTOWG/ PET328_2021_Class* repository.

GETTING STARTED WITH PYTHON

Repeated Execution
while... loops

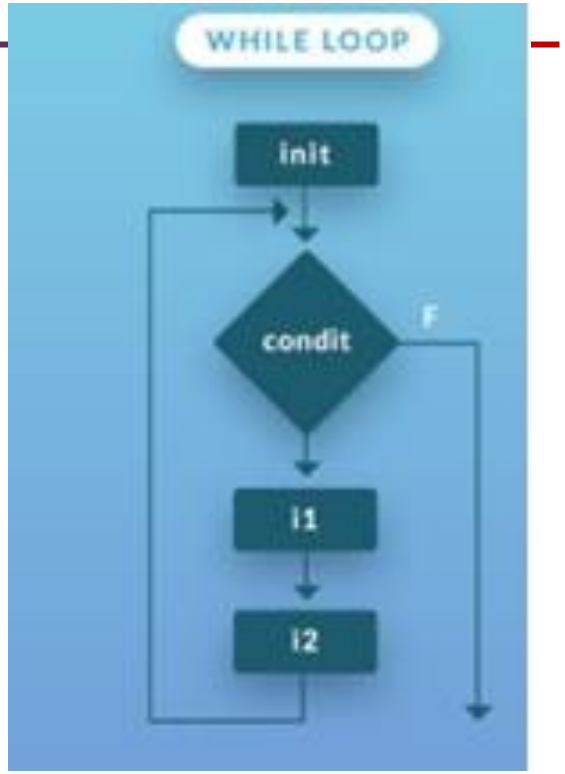


- Indefinite loops are implemented with 'while' loops.
- Unlike 'for' loops, a definite list of items to be acted upon is not presented to 'while' loops.
- Rather, 'while' loops performs a given block of statement(s) until a specified condition is evaluated to be False; then it stops.
- The loop runs if the specified condition is True, just like an 'if' statement.
- However, unlike an 'if' statement, the 'while' loop repeats the block of statement(s).

GETTING STARTED WITH PYTHON

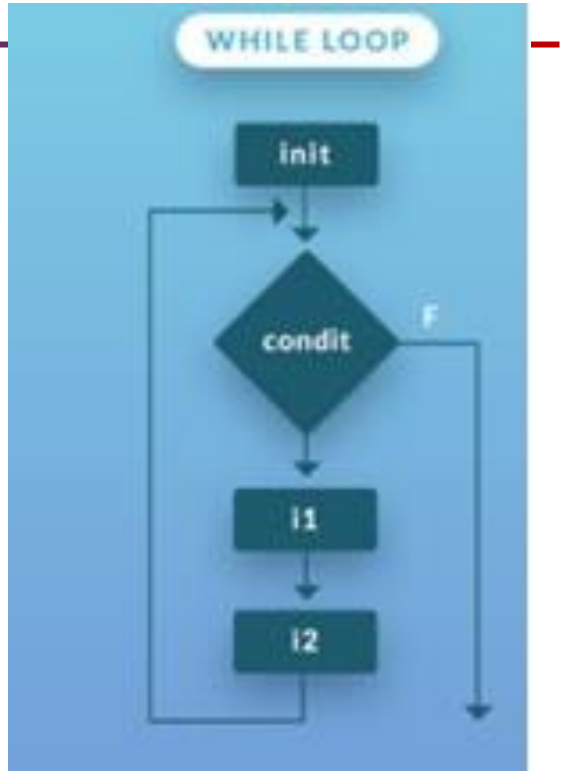
Repeated Execution
while... loops

- The execution of 'while' loops goes thus:
  - Evaluate the specified condition
  - If condition is False, exit the loop and continue with statements(s) outside (below) the loop.
  - Execute the block of statement(s) if condition is true and return to evaluation step



GETTING STARTED WITH PYTHON

Repeated Execution
while... loops

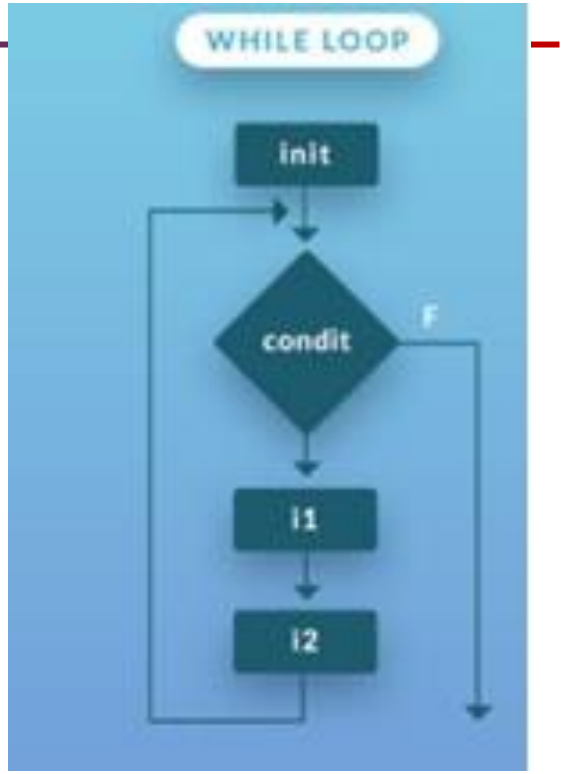


- The 'while' loop condition is constructed as Boolean expression.
- Typically, the condition involves an iterator variable that is normally initialized outside (before) the loop and gets updated in the loop (during each iteration).
- It is by such changes (updates) to the value of the iterator variable that the condition would eventually turn False and the loop gets terminated.

GETTING STARTED WITH PYTHON

Repeated Execution
while... loops

- When the iteration variable is not specified or not updated, the loop runs forever – infinite loops.
-
- Reading Assignment:
  - Read on Infinite Loops, 'break' and 'continue' statements in the textbook recommended for this course.



GETTING STARTED WITH PYTHON

Repeated Execution
while... Loops

Petroleum engineering applications

- The 'while' loops finds applications in cases where a task need to be performed for a series of dynamically-changing values of a variable (like pressure) until that variable violates a specified threshold value.

- For instance, in scripting an undersaturated oil reservoir simulator, computations are to be performed for series of decreasing pressure values until pressure reduces to a value below bubble-point pressure.

GETTING STARTED WITH PYTHON

Repeated Execution
while... Loops

Petroleum engineering applications

- See a trivial example here.

```
>>> current_pressure = 4000
>>> bubble_pressure = 2800
>>> while current_pressure > bubble_pressure:
        print('Pressure {0} is still undersaturated, perform computations'.format(current_pressure))
        current_pressure = current_pressure - 200


Pressure 4000 is still undersaturated, perform computations
Pressure 3800 is still undersaturated, perform computations
Pressure 3600 is still undersaturated, perform computations
Pressure 3400 is still undersaturated, perform computations
Pressure 3200 is still undersaturated, perform computations
Pressure 3000 is still undersaturated, perform computations
```

GETTING STARTED WITH PYTHON

Repeated Execution
while... Loops

Petroleum engineering applications

- Updating oil formation volume factor (Bo), for undersaturated reservoir simulation.

Recall that, the analytical equation for computing Bo, at pressures above bubble point is given as: (see Slide 75 – 76)

$$B_o = B_{ob} e^{[c_o(P_b - P)]} \quad ---- (2.37)$$

- There is a need to write statement that would not just compute Bo at a single pressure value, but at series of decreasing pressure values until pressure decreases to bubble point pressure, at which point, the simulation must be terminated.

GETTING STARTED WITH PYTHON

Repeated Execution
while... Loops

Petroleum engineering applications

- Updating oil formation volume factor (Bo), for undersaturated reservoir simulation.

```
while p > pb:
    bo = bob*(math.exp(co*(pb-p)))
    print('The value of oil FVF at {0:.2f} psi is {1:.4f}'.format(p, bo))
    p = p - pressure_step

# continuing after the 'while' block
print('Bubble-point pressure reached! End of simulation')
```

The full script for this computation is available here.

GETTING STARTED WITH PYTHON

Repeated Execution
while... Loops

Petroleum engineering applications

- Updating oil formation volume factor (Bo), for undersaturated reservoir simulation.

```
 RESTART: C:/Users/TTOWG/645/1 karia def/2. CU/CU Cours
PET328_2021_Class/demo_oil_fvf_updator.py
What is the value of reservoir initial pressure?4000
What is the value of reservoir bubble-point pressure?3330
What is the value of oil FVF at bubble-point pressure?1.2511
What is the value of oil compressibility?0.0000113
By how much should pressure be decremented?50
The value of oil FVF at 4000.00 psi is 1.2417
The value of oil FVF at 3950.00 psi is 1.2424
The value of oil FVF at 3900.00 psi is 1.2431
The value of oil FVF at 3850.00 psi is 1.2438
The value of oil FVF at 3800.00 psi is 1.2445
The value of oil FVF at 3750.00 psi is 1.2452
The value of oil FVF at 3700.00 psi is 1.2459
The value of oil FVF at 3650.00 psi is 1.2466
The value of oil FVF at 3600.00 psi is 1.2473
The value of oil FVF at 3550.00 psi is 1.2480
The value of oil FVF at 3500.00 psi is 1.2487
The value of oil FVF at 3450.00 psi is 1.2494
The value of oil FVF at 3400.00 psi is 1.2501
The value of oil FVF at 3350.00 psi is 1.2508
Bubble-point pressure reached! End of simulation
```