Python for Security

Python 3
Crash Course

by

Dr. Mona Fouad

mfouad@nti.sci.eg

2021-2022

NTI
Four-Months Track

Course Outline

- First Day—Python Crash Course
  - Programming Overview
  - Python Installations
  - Python Programming Concepts and Hands-on

Course Outline (cont.)

- Second Day—Cryptography using Python
  - Cryptography Overview
  - Caesar Ciphering
  - Modular Arithmetic
    - Modular Inverse
    - Greatest Common Divisor
  - Affine Ciphering
  - Attacking Encrypted messages

Course Outline (cont.)

- Thirsd Day—Advanced Topics
  - Vigenère and one-time pad ciphers
  - Base64 CODEC
  - Hashing and password verification
  - DES and AES Cryptography
  - Public Key and Digital Signature
- References

Python for Security

Programming Overview

Contents

- What is computer programming?
- Interpreter vs. Compiler
- Computer languages’ compositions
- Python versions
- Pythons implementations
- Python programming tools

Computer Languages

- Machine language vs. High level languages
- A computer high-level language is formed of alphabets, lexis (words), syntax, and semantics
- Source code—A program written with a high-level language
- Interpretation/Compilation—Transforming a program from a high-level programming language into machine language

Interpretation vs. Compilation

	Compilation	**Interpretation**
Advantages	✓the execution is usually faster; ✓only the user has to have the compiler – the end-user doesn’t need to run the code; ✓the translated code is stored using machine language – as it is very hard to understand it, your own inventions and programming tricks are likely to remain your secret.	✓you can run the code as soon as you complete it – there are no additional phases of translation; ✓the code is stored using programming language, not machine language - this means that it can be run on computers using different machine languages; you don't compile your code separately for each different architecture.
Disadvantages	✕ the compilation itself may be a very time-consuming process – you may not be able to run your code immediately after making an amendment; ✕ you have to have as many compilers as hardware platforms you want your code to be run on.	✕ don't expect interpretation to ramp up your code to high speed - your code will share the computer's power with the interpreter, so it can't be really fast; ✕ both you and the end user have to have the interpreter to run your code.

Python Language

- Python is a widely-used, interpreted, object-oriented, and high-level programming language with dynamic semantics.
- If you want to program in Python, you'll need the Python interpreter.
- Fortunately, Python is free.
- Languages designed to be utilized in the interpretation manner are often called *scripting* *languages*.
- Python was ***conceived*** in the late 1980s by ***Guido van Rossum*** at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC programming language.
- Python name is come from a BBC television comedy sketch series called Monty Python's Flying Circus.

Python: A HOBBY Programming Project

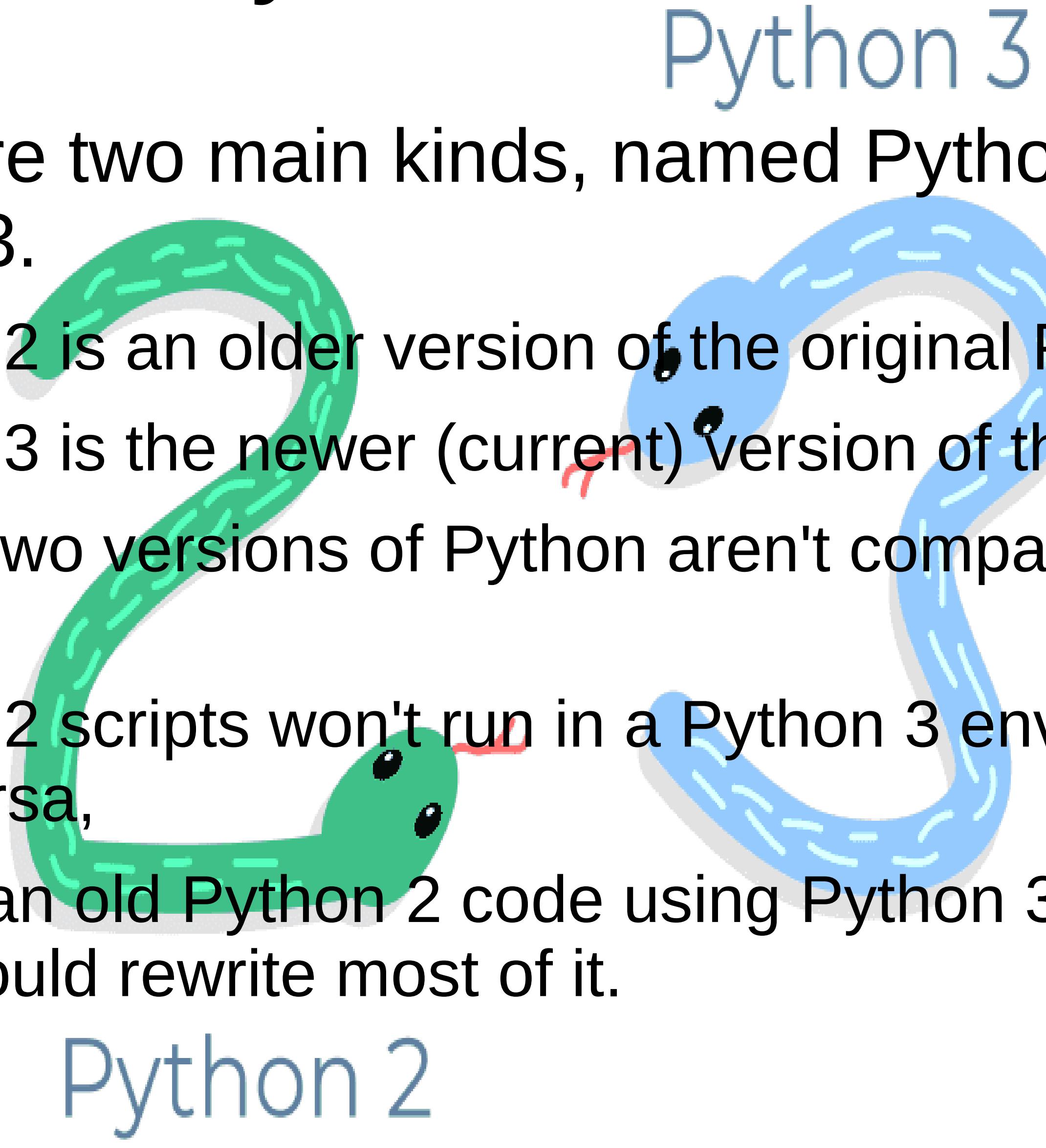
“In December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office (...) would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).” ***Guido van Rossum***

- Python goals In 1999, Guido van Rossum defined his goals for Python:
  - an easy and intuitive language just as powerful as those of the major competitors;
  - open source, so anyone can contribute to its development;
  - code that is as understandable as plain English;
  - suitable for everyday tasks, allowing for short development times.
- About 20 years later, it is clear that all these intentions have been fulfilled.

Computer Language Compositions

- Alphabets—a set of symbols used to build words of a certain language (e.g., the Latin alphabet for English, the Cyrillic alphabet for Russian, Kanji for Japanese, and so on)
- Lexis—(aka a dictionary) a set of words the language offers its users (e.g., the word "computer" comes from the English language dictionary, while "cmoptrue" doesn't; the word "chat" is present both in English and French dictionaries, but their meanings are different)
- Syntax—a set of rules (formal or informal, written or felt intuitively) used to determine if a certain string of words forms a valid sentence (e.g., "I am a python" is a syntactically correct phrase, while "I a python am" isn't)
- Semantics—a set of rules determining if a certain phrase makes sense (e.g., "I ate a doughnut" makes sense, but "A doughnut ate me" doesn't)

Python Versions



There are two main kinds, named Python 2 and Python 3.

- Python 2 is an older version of the original Python.
- Python 3 is the newer (current) version of the language.
- These two versions of Python aren't compatible with each other.
- Python 2 scripts won't run in a Python 3 environment and vice versa,
- To run an old Python 2 code using Python 3 interpreter, you should rewrite most of it.

Python Implementations

- The traditional implementation of Python, called CPython
- Guido van Rossum used the "C" programming language to implement **Cpython**.
- **Jython** ("J" is for "Java") can communicate with existing Java infrastructure effectively. Till now, Jython implementation follows Python 2 standards only.
- **PyPy** is a Python within a Python. It named RPython (Restricted Python). It is a tool for people developing Python. It is compatible with Python 3.
- **MicroPython** is an efficient open source software implementation of Python 3 that is optimized to run on microcontrollers.

Python Programming Tools

To start programming, the following tools are needed:

- An ***editor*** for writing the code (it should have some special features, not available in simple tools); this dedicated editor will give more than the standard OS editor;
- A ***console*** to launch the written code and stop it forcibly when it gets out of control;
- A ***debugger***, able to launch your code step-by-step, which will allow you to inspect it at each moment of execution.
- An ***IDLE*** Integrated Development and Learning Environment.

Python for Security

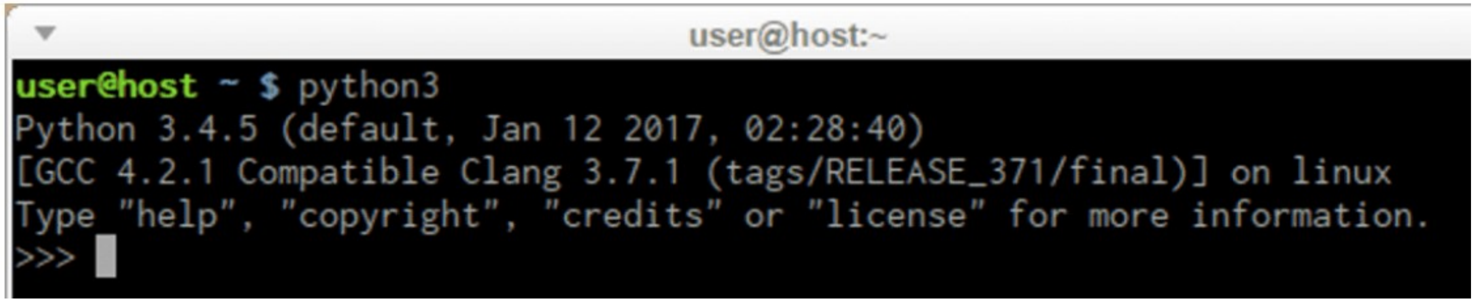
Python Installation

Contents

- Python 3 installation
- Integrated Development Environment IDE
- Anaconda installation
- PyCharm as Python editor
- Visual Studio Code (free visual studio)

Python 3

- How to get Python 3 Interpreter ?
- There are several ways to get your own copy of Python 3, depending on the operating system you use.
  - Linux users most probably have Python already installed - this is the most likely scenario, as Python's infrastructure is intensively used by many Linux OS components.
  - If you're a Linux user, open the terminal/console, and type: python3
    - You may get something like that:



user@host:~

```
user@host ~ $ python3
Python 3.4.5 (default, Jan 12 2017, 02:28:40)
[GCC 4.2.1 Compatible Clang 3.7.1 (tags/RELEASE_371/final)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python 3 (cont.)

- All Linux or non-Linux users can download and install a copy at https://www.python.org/downloads/ and then you can choose your platform for installation

✔ Anaconda is another option to install Python 3
  - Anaconda is recommended: https://docs.anaconda.com/anaconda/install/
  - Anaconda is a platform-agnostic, so you can use it whether you are on Windows, mac-OS, or Linux.
  - Anaconda is free and easy to install, and it offers free community support.

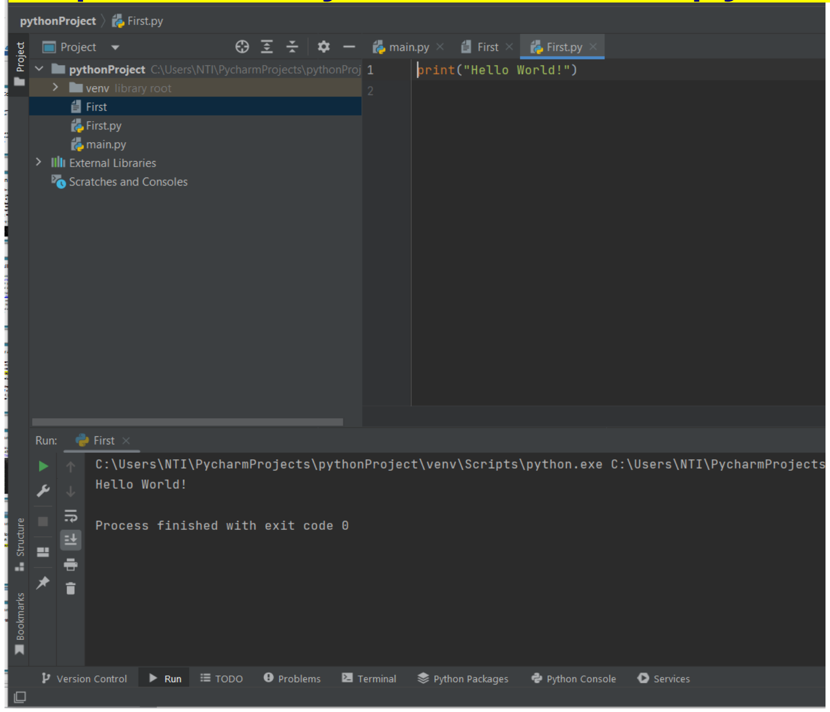
Integrated Development and Learning Environment

- Every Python installation comes with an Integrated Development and Learning Environment, which you’ll see shortened to **IDLE** or even IDE.
- Python IDLE comes included in Python installations on Windows and Mac.
- If you’re a Linux user, then you should be able to find and download Python IDLE using your package manager.
- Once you’ve installed it, you can then use Python IDLE as an interactive interpreter or as a file editor.

PyCharm

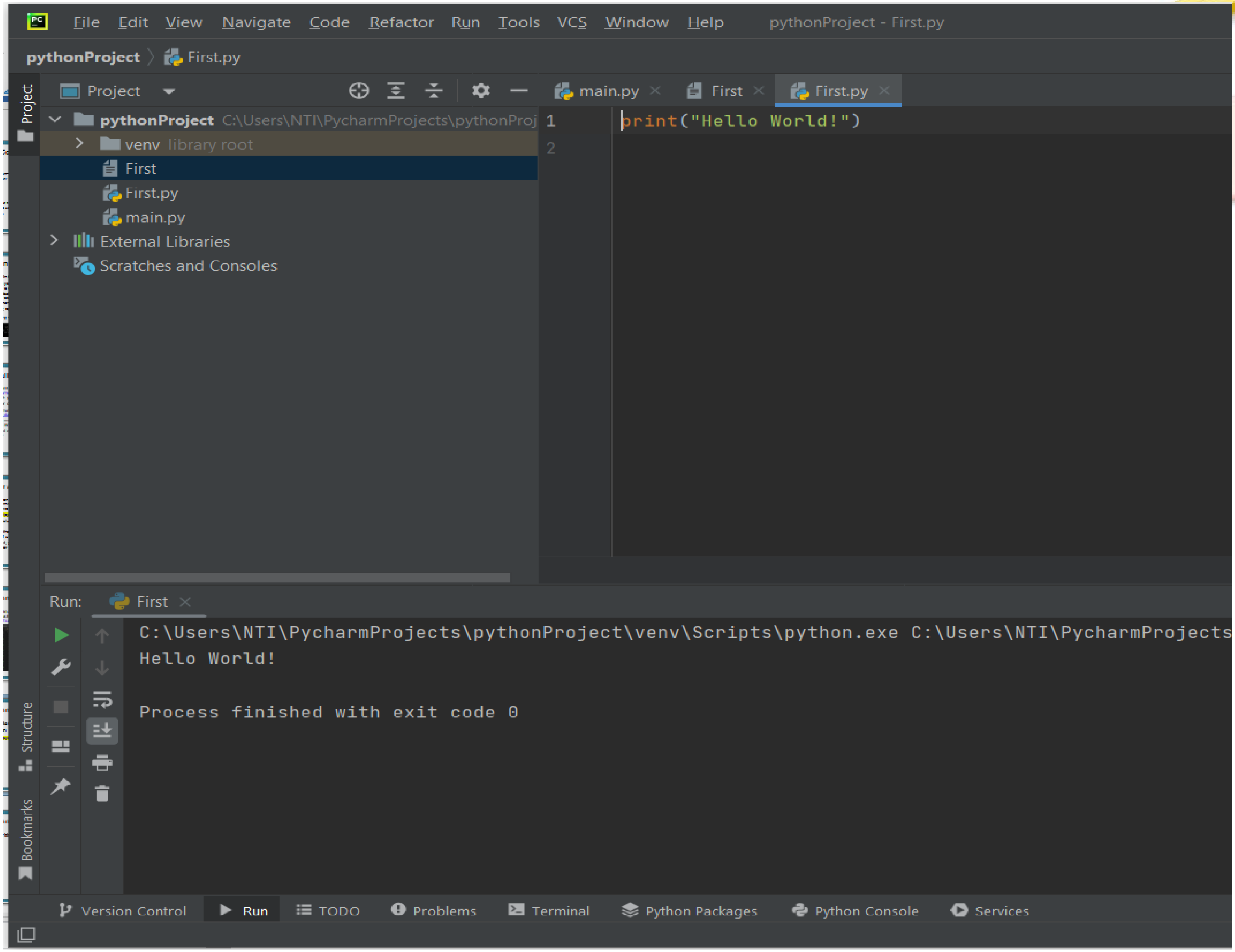
- Another Python editor option is PyCharm
- Community version is free

https://www.jetbrains.com/pycharm/download/#section=windows



```
print("Hello World!")
```

```
C:\Users\NTI\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\NTI\PycharmProjects
Hello World!

Process finished with exit code 0
```

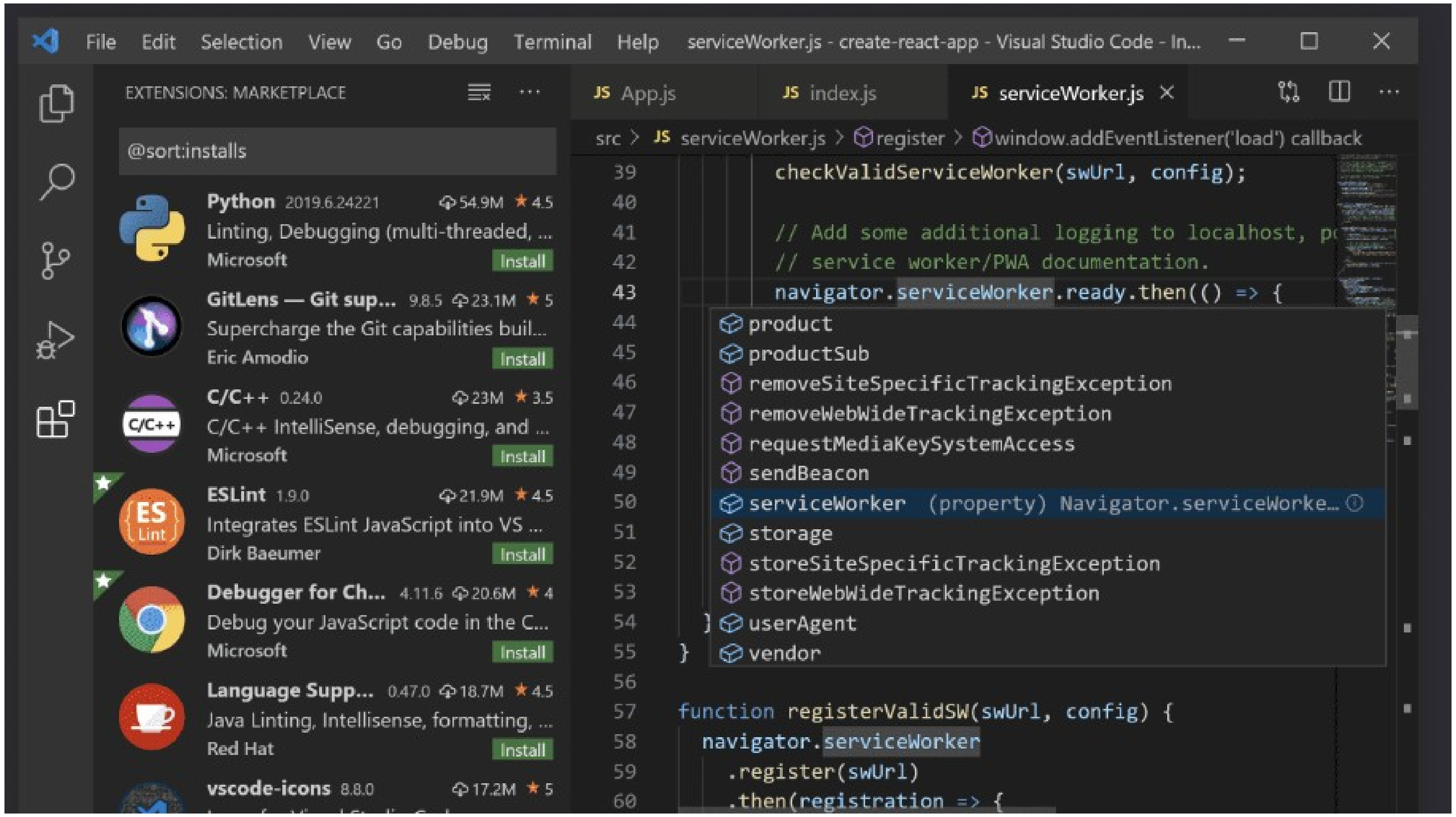


```python
print("Hello World!")
```

```
C:\Users\NTI\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\NTI\PycharmProjects
Hello World!

Process finished with exit code 0
```

Visual Studio Code (VSCode)

- VSCode is a free code editor (such visual studio code VSCode at)https://code.visualstudio.com/


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

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

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

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

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

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

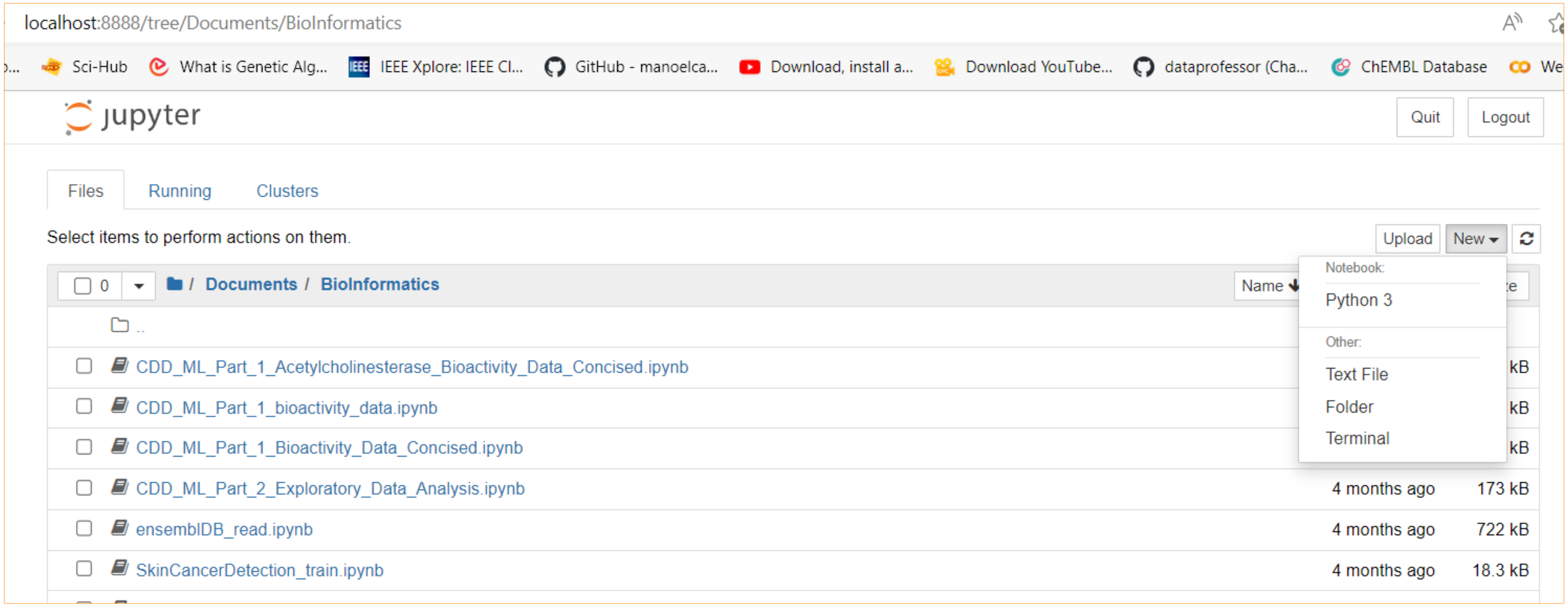
Language Supp... 0.47.0 18.7M ★4.5
Java Linting, Intellisense, formatting, ...
Red Hat Install

vscode-icons 8.8.0 17.2M ★5

```
      checkValidServiceWorker(swUrl, config);

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

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

Jupyter Notebook

✔ If you installed Anaconda ***successfully***, you may work with **Jupyter Notebook** directly:


localhost:8888/tree/Documents/BioInformatics

Sci-Hub | What is Genetic Alg... | IEEE Xplore: IEEE Cl... | GitHub - manoelca... | Download, install a... | Download YouTube... | dataprofessor (Cha... | ChEMBL Database | We

jupyter — Quit — Logout

Files | Running | Clusters

Select items to perform actions on them.

Upload | New ▾

0 ▾ / Documents / BioInformatics — Name

..

CDD_ML_Part_1_Acetylcholinesterase_Bioactivity_Data_Concised.ipynb — kB

CDD_ML_Part_1_bioactivity_data.ipynb — kB

CDD_ML_Part_1_Bioactivity_Data_Concised.ipynb — kB

CDD_ML_Part_2_Exploratory_Data_Analysis.ipynb — 4 months ago — 173 kB

ensemblDB_read.ipynb — 4 months ago — 722 kB

SkinCancerDetection_train.ipynb — 4 months ago — 18.3 kB

Jupyter Notebook (cont.)

- Test the installed environment
  - Open anaconda terminal and type ***Jupyter notebook***
- Write your first script:

  print(“Hello World!”)
- ***Let us work a bit with Jupyter notebook cells***
  - Create new notebook or edit an existing one
  - Edit and run cell(s)
  - Name and save the created notebook
  - Cell could be for python codes or text
  - Examine this code *print(“Hello World”)*

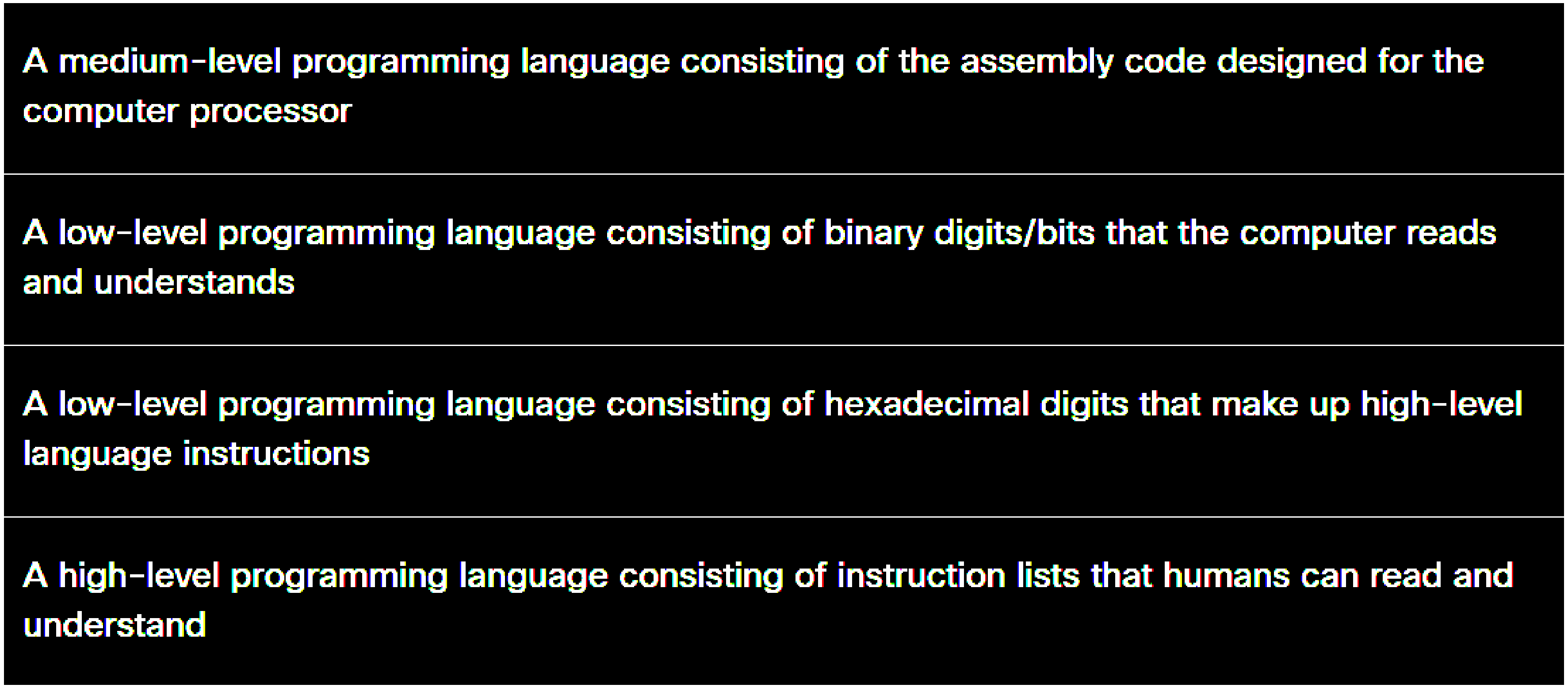
Python for Security

Knowledge Check

Quiz 1

Cisco Networking Academy

- What is a machine code?



A medium-level programming language consisting of the assembly code designed for the computer processor

A low-level programming language consisting of binary digits/bits that the computer reads and understands

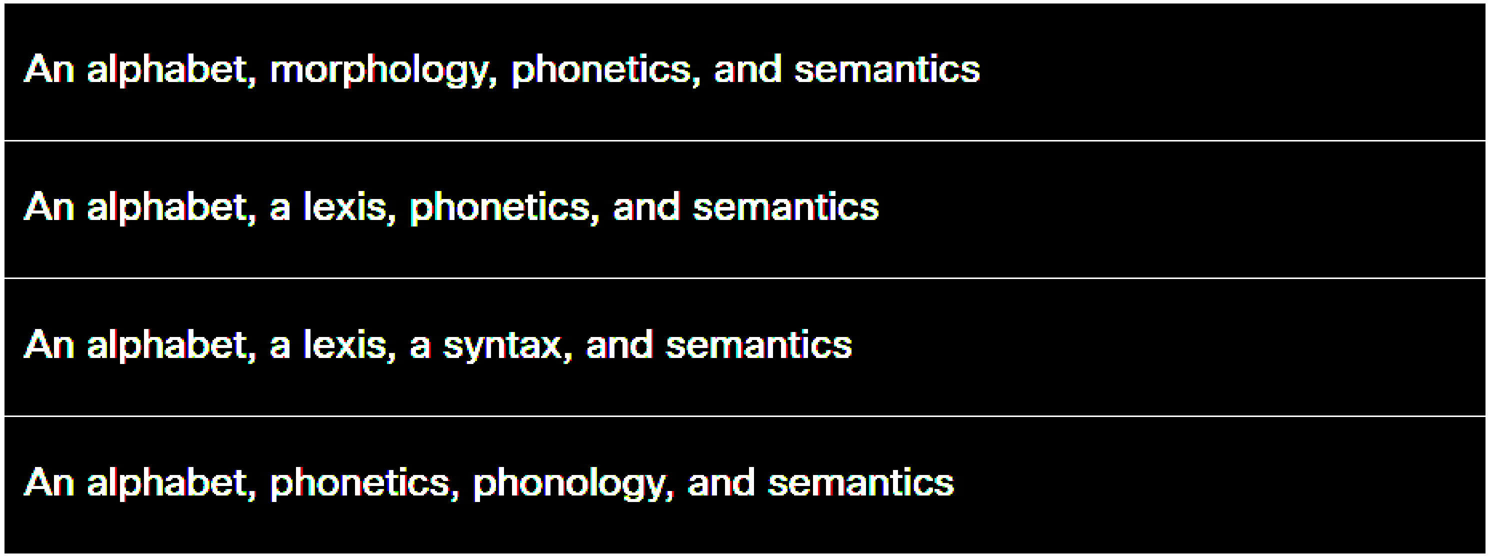
A low-level programming language consisting of hexadecimal digits that make up high-level language instructions

A high-level programming language consisting of instruction lists that humans can read and understand

Quiz 2

Cisco Networking Academy

- What are the four fundamental elements that make a language?



An alphabet, morphology, phonetics, and semantics

An alphabet, a lexis, phonetics, and semantics

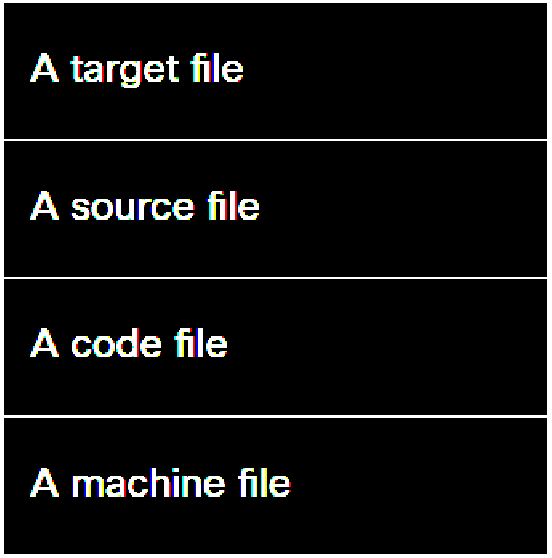
An alphabet, a lexis, a syntax, and semantics

An alphabet, phonetics, phonology, and semantics

Quiz 3

Cisco Networking Academy

- What do you call a file containing a program written in a high-level programming language?



A target file

A source file

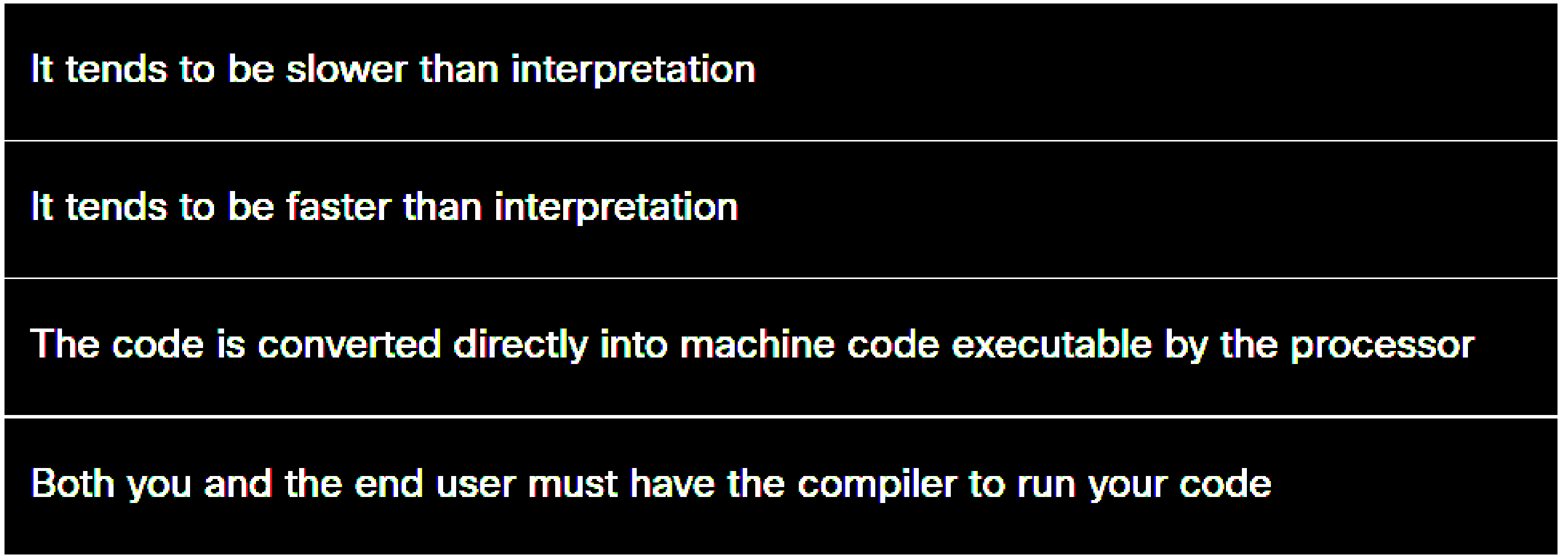
A code file

A machine file

Quiz 4

Cisco Networking Academy

- What is true about compilation? (Select two answers)



It tends to be slower than interpretation

It tends to be faster than interpretation

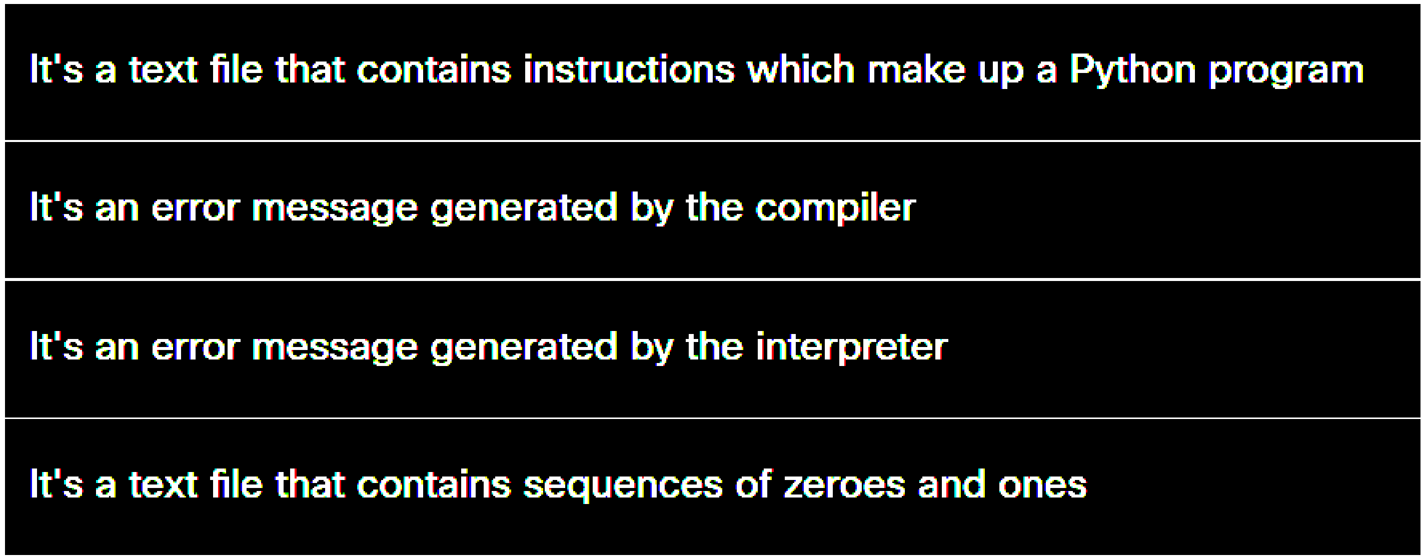
The code is converted directly into machine code executable by the processor

Both you and the end user must have the compiler to run your code

Quiz 5

Cisco Networking Academy

- What is the best definition of a Python script?

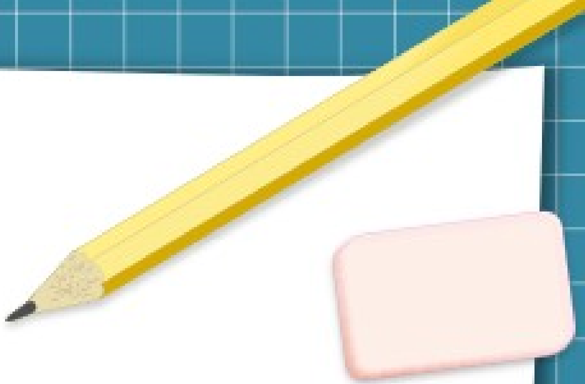


It's a text file that contains instructions which make up a Python program

It's an error message generated by the compiler

It's an error message generated by the interpreter

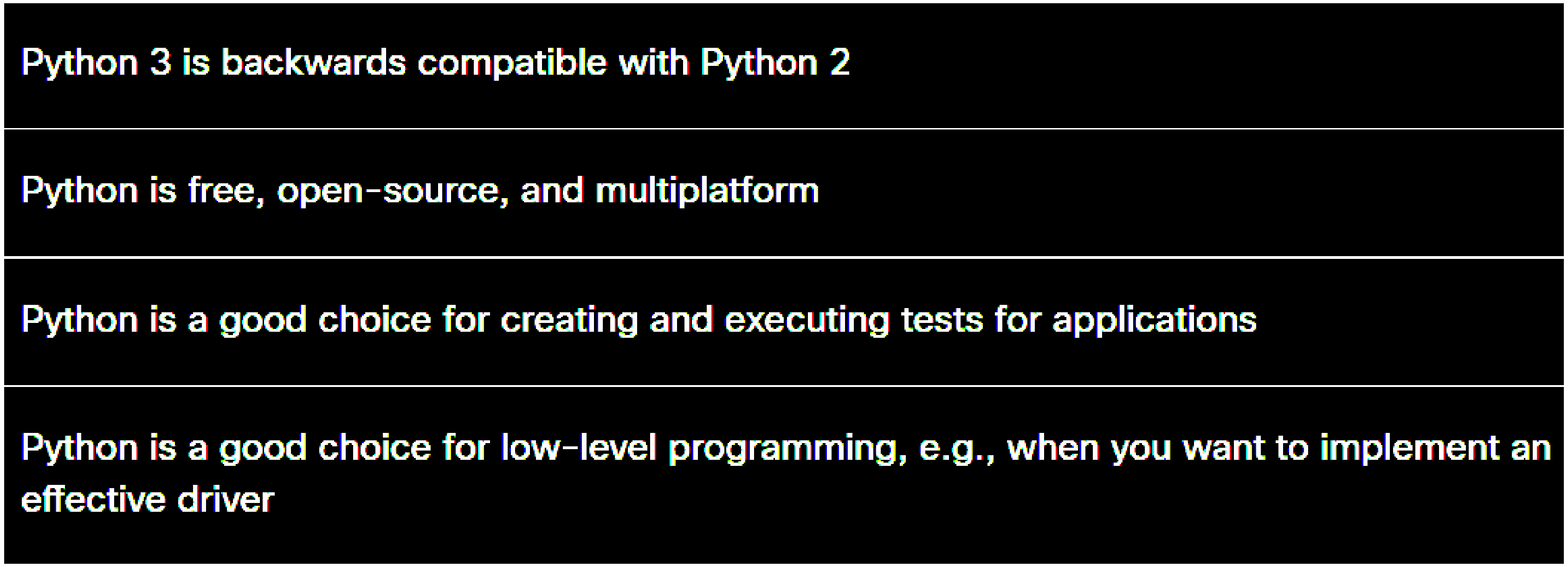
It's a text file that contains sequences of zeroes and ones



Quiz 6

Cisco Networking Academy

- Select the true statements. (Select two answers)



Python 3 is backwards compatible with Python 2

Python is free, open-source, and multiplatform

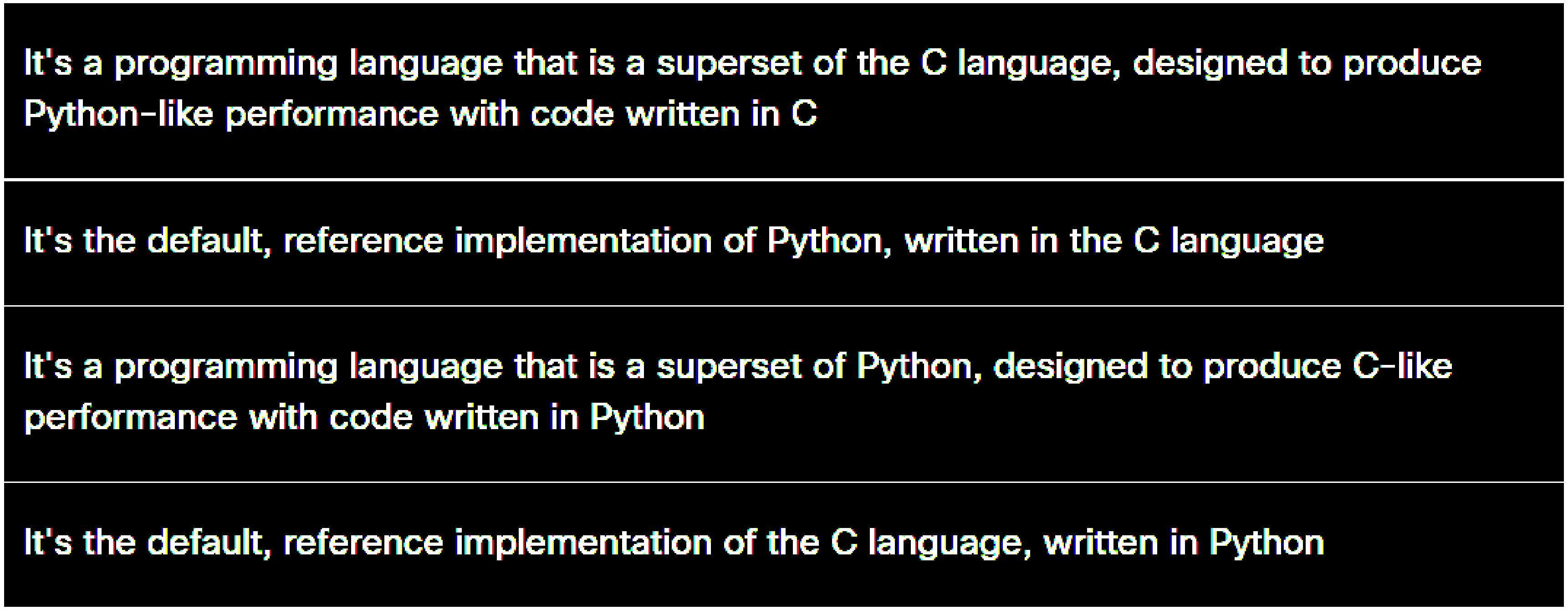
Python is a good choice for creating and executing tests for applications

Python is a good choice for low-level programming, e.g., when you want to implement an effective driver

Quiz 7

Cisco Networking Academy

- What is CPython?



It's a programming language that is a superset of the C language, designed to produce Python-like performance with code written in C

It's the default, reference implementation of Python, written in the C language

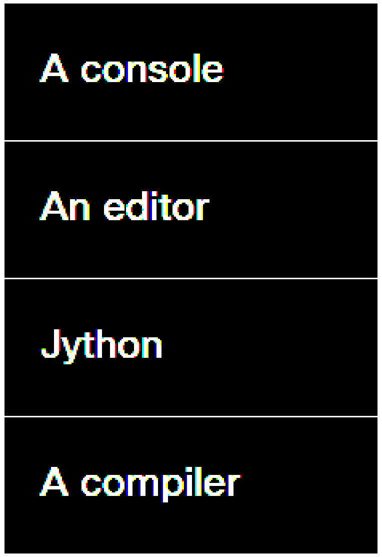
It's a programming language that is a superset of Python, designed to produce C-like performance with code written in Python

It's the default, reference implementation of the C language, written in Python

Quiz 8

Cisco Networking Academy

- What do you call a command-line interpreter which lets you interact with your OS and execute Python commands and scripts?



A console

An editor

Jython

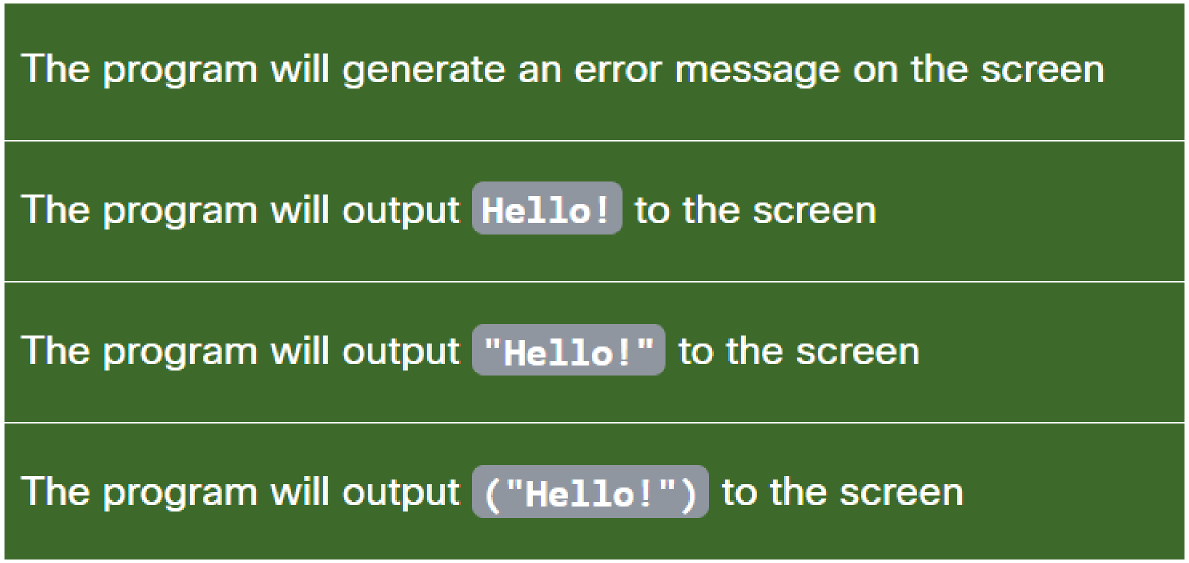
A compiler

Quiz 9

Cisco Networking Academy

- What is the expected behavior of the following program?

```
print(“Hello!”)
```



The program will generate an error message on the screen

The program will output `Hello!` to the screen

The program will output `"Hello!"` to the screen

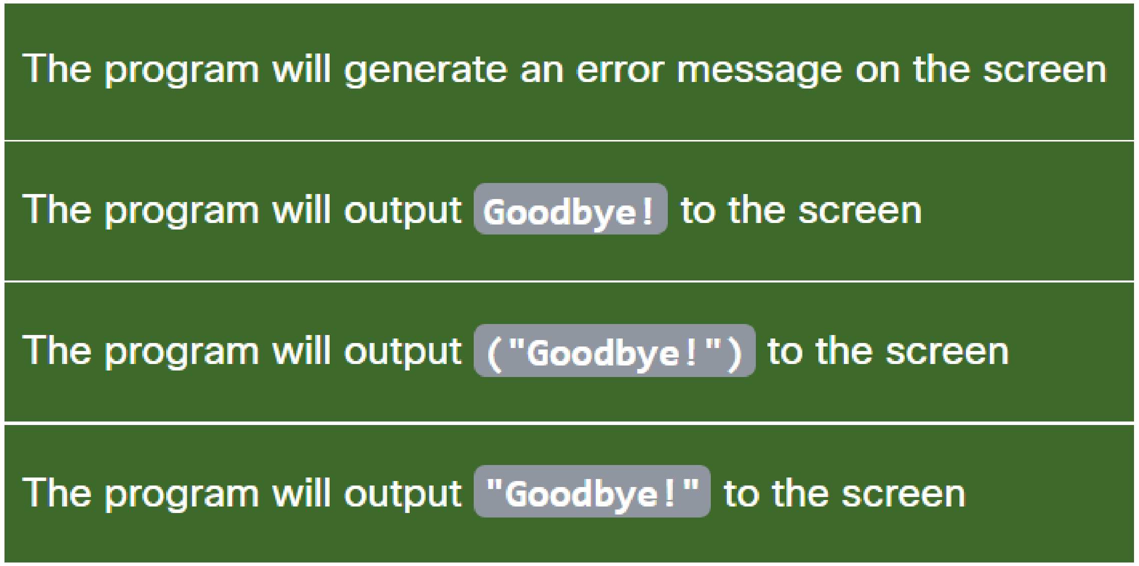
The program will output `("Hello!")` to the screen

Quiz 10

Cisco Networking Academy

- What is the expected behavior of the following program?

```
prin("Goodbye!")
```



- The program will generate an error message on the screen
- The program will output `Goodbye!` to the screen
- The program will output `("Goodbye!")` to the screen
- The program will output `"Goodbye!"` to the screen

Python for Security

Python Programming Concepts and Hands-on

Contents

Guide for programming Activities

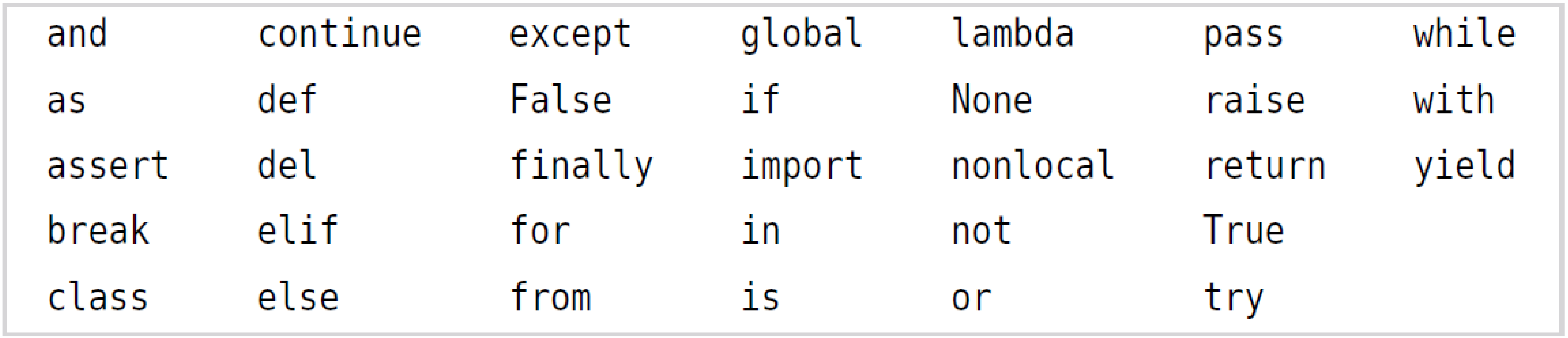
Right Now

- Invoke the Python prompt (execute the Windows shell, or Linux terminal) → *python3*
- Or, create a new notebook from jupyter anaconda environment → **>** *jupyter notebook*

• During this course, invoke the python prompt (or jupyter notebook) daily, once you enter the class room.

Data Types and Variables

- **User Identifiers**—Name of objects you create is called identifier. A valid Python identifier is a nonempty sequence of characters of any length that consists of a “start character” and zero or more “continuation characters”.
- Some **Python Keywords**

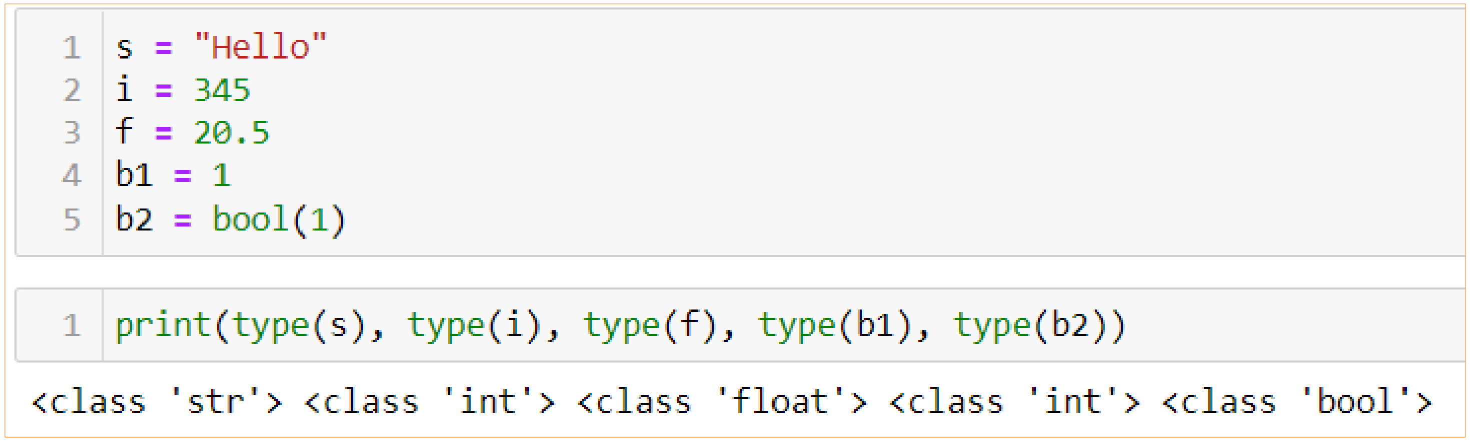


`and`	`continue`	`except`	`global`	`lambda`	`pass`	`while`
`as`	`def`	`False`	`if`	`None`	`raise`	`with`
`assert`	`del`	`finally`	`import`	`nonlocal`	`return`	`yield`
`break`	`elif`	`for`	`in`	`not`	`True`	
`class`	`else`	`from`	`is`	`or`	`try`	

- User identifier should not contradict with Python keywords or identifiers.

Data Types and Variables (cont.)

- In Python, you don't need to specify the variable data type.
- Just use the variable, and Python will detect its type.



```python
s = "Hello"
i = 345
f = 20.5
b1 = 1
b2 = bool(1)
```

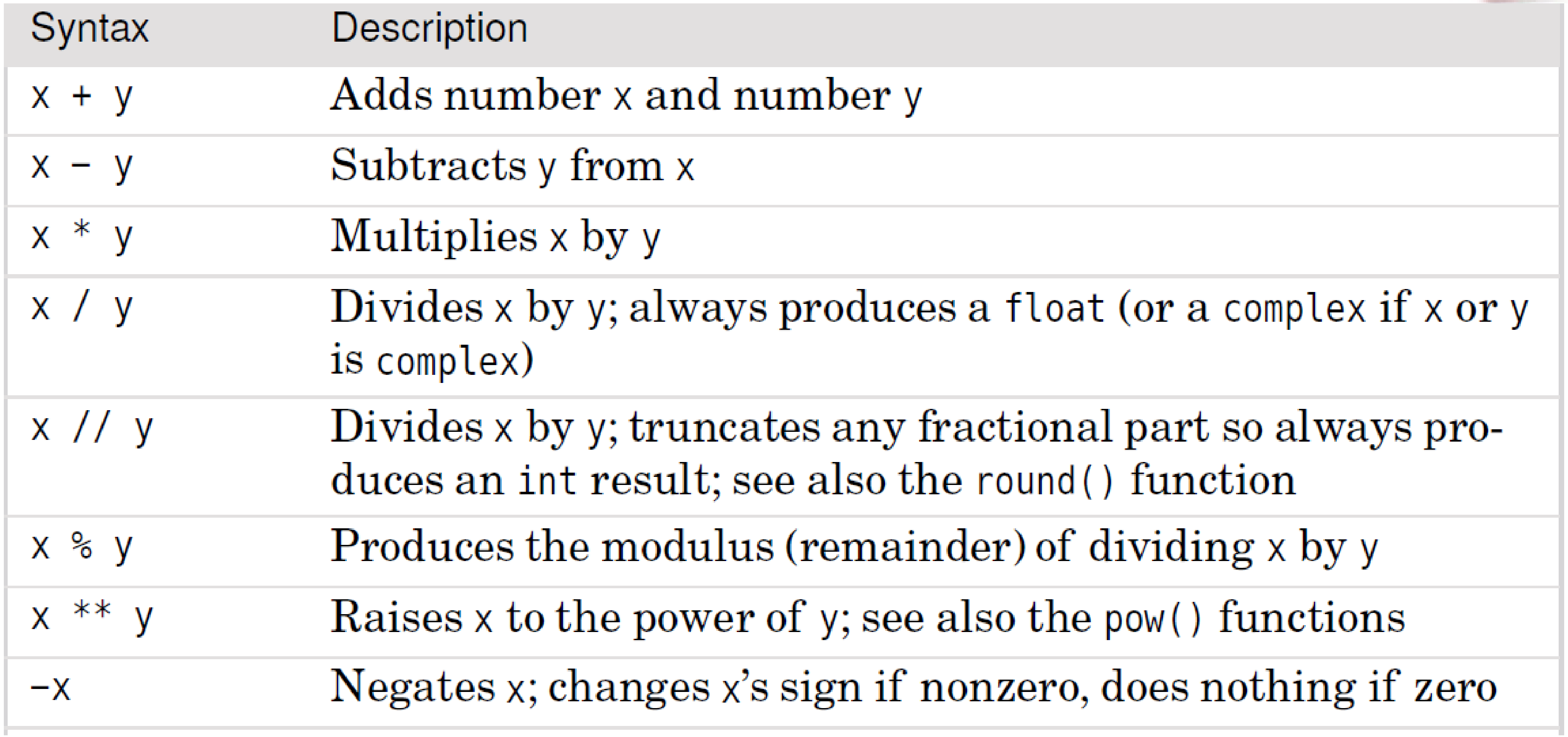
```python
print(type(s), type(i), type(f), type(b1), type(b2))
```

```
<class 'str'> <class 'int'> <class 'float'> <class 'int'> <class 'bool'>
```

Integers, Boolean, and Float

- Integers—those numbers in the form of digits (ex. 36464524, 10, 40000, 34725, …)
- Boolean—a one digit number, either 1 or 0 (representing True or False)
- Float—those numbers in the form of digit and decimal (ex. 0.0, 0.765, 34.12, 75755.53, ...)

Numerical Operators



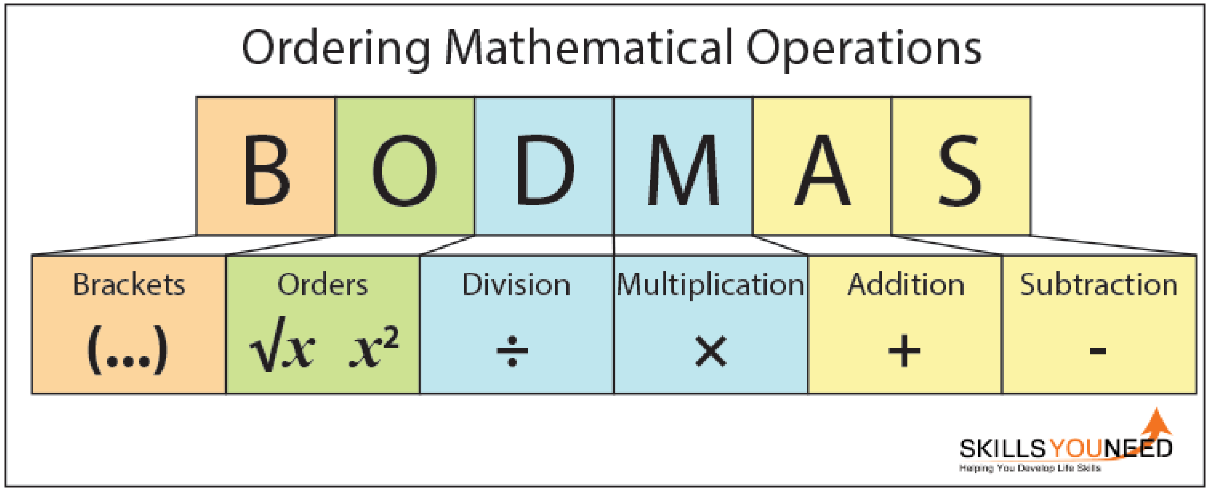
Syntax	Description
`x + y`	Adds number `x` and number `y`
`x - y`	Subtracts `y` from `x`
`x * y`	Multiplies `x` by `y`
`x / y`	Divides `x` by `y`; always produces a `float` (or a `complex` if `x` or `y` is `complex`)
`x // y`	Divides `x` by `y`; truncates any fractional part so always produces an `int` result; see also the `round()` function
`x % y`	Produces the modulus (remainder) of dividing `x` by `y`
`x ** y`	Raises `x` to the power of `y`; see also the `pow()` functions
`-x`	Negates `x`; changes `x`'s sign if nonzero, does nothing if zero

Orders of the Mathematical Operations

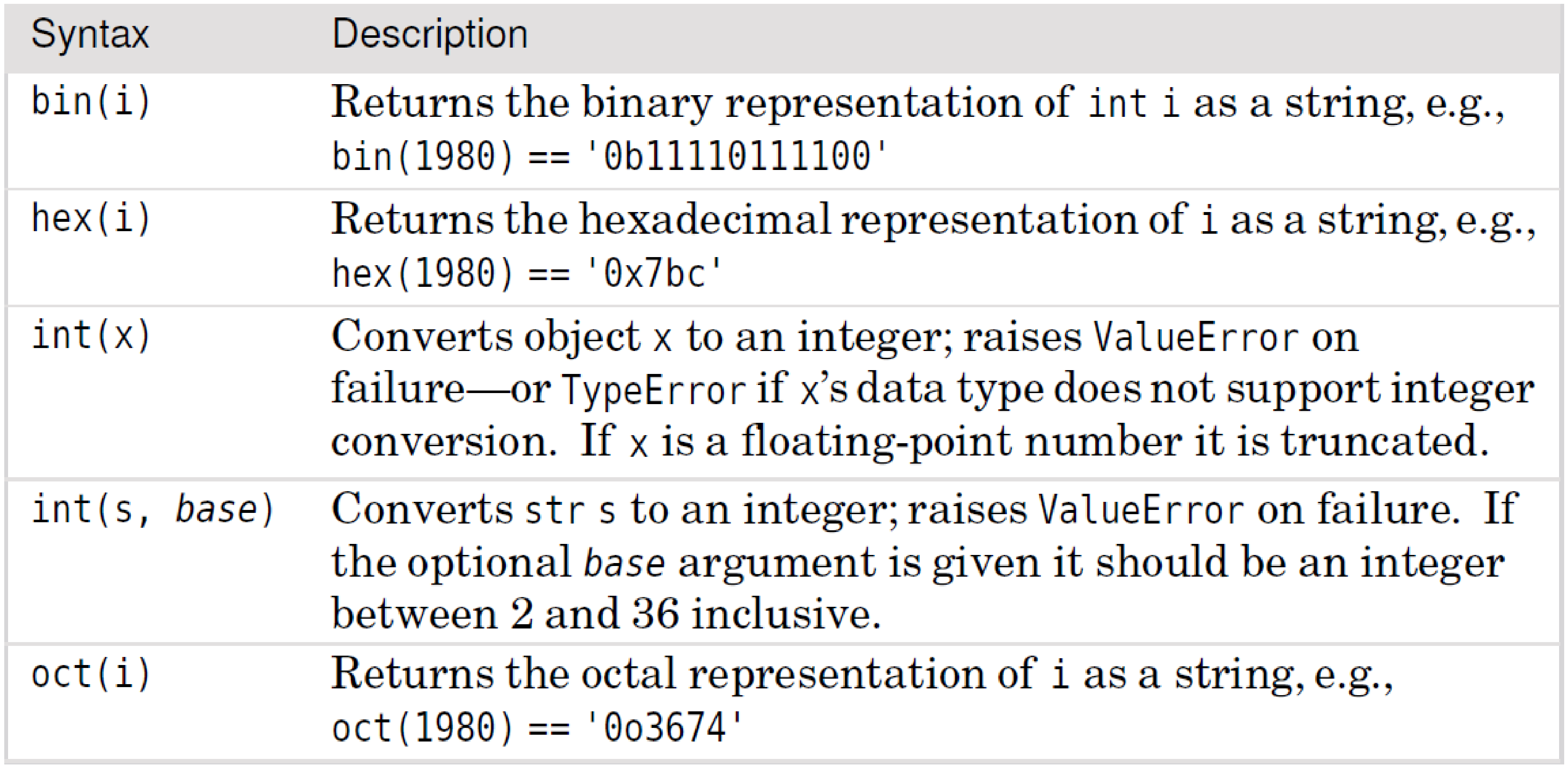
- Mathematical operations

30 + (10*3 – 34/6)**0.5 / 50*23

- Order of mathematical operations

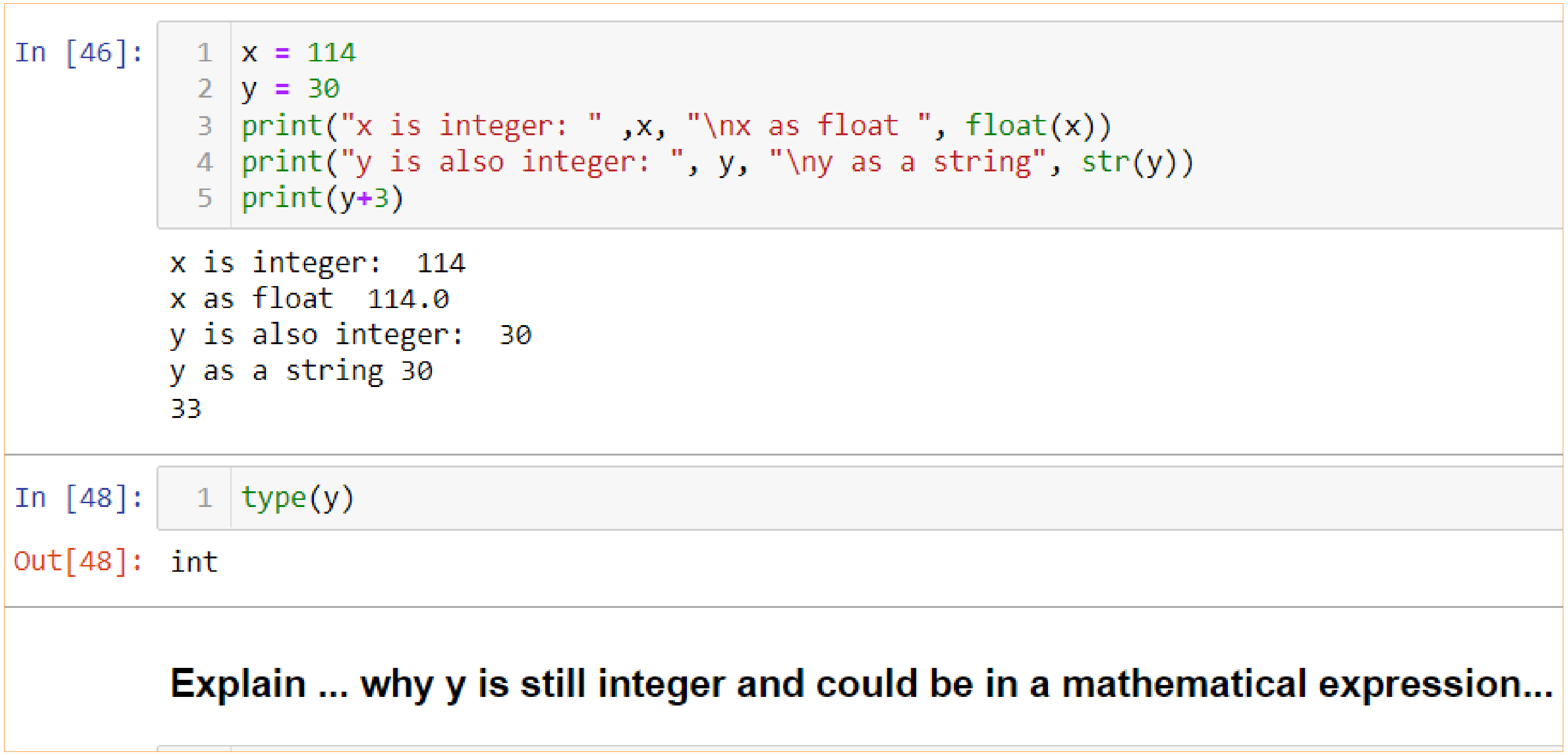


Integer Conversions (Casting)



Syntax	Description
`bin(i)`	Returns the binary representation of `int i` as a string, e.g., `bin(1980) == '0b11110111100'`
`hex(i)`	Returns the hexadecimal representation of `i` as a string, e.g., `hex(1980) == '0x7bc'`
`int(x)`	Converts object `x` to an integer; raises `ValueError` on failure—or `TypeError` if `x`'s data type does not support integer conversion. If `x` is a floating-point number it is truncated.
`int(s, *base*)`	Converts `str s` to an integer; raises `ValueError` on failure. If the optional *base* argument is given it should be an integer between 2 and 36 inclusive.
`oct(i)`	Returns the octal representation of `i` as a string, e.g., `oct(1980) == '0o3674'`

Casting Examples



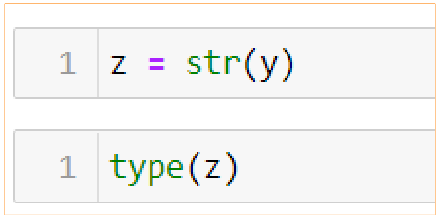
```
In [46]:
x = 114
y = 30
print("x is integer: " ,x, "\nx as float ", float(x))
print("y is also integer: ", y, "\ny as a string", str(y))
print(y+3)
```

```
x is integer:  114
x as float  114.0
y is also integer:  30
y as a string 30
33
```

```
In [48]:
type(y)
```

```
Out[48]: int
```

Explain ... why y is still integer and could be in a mathematical expression...

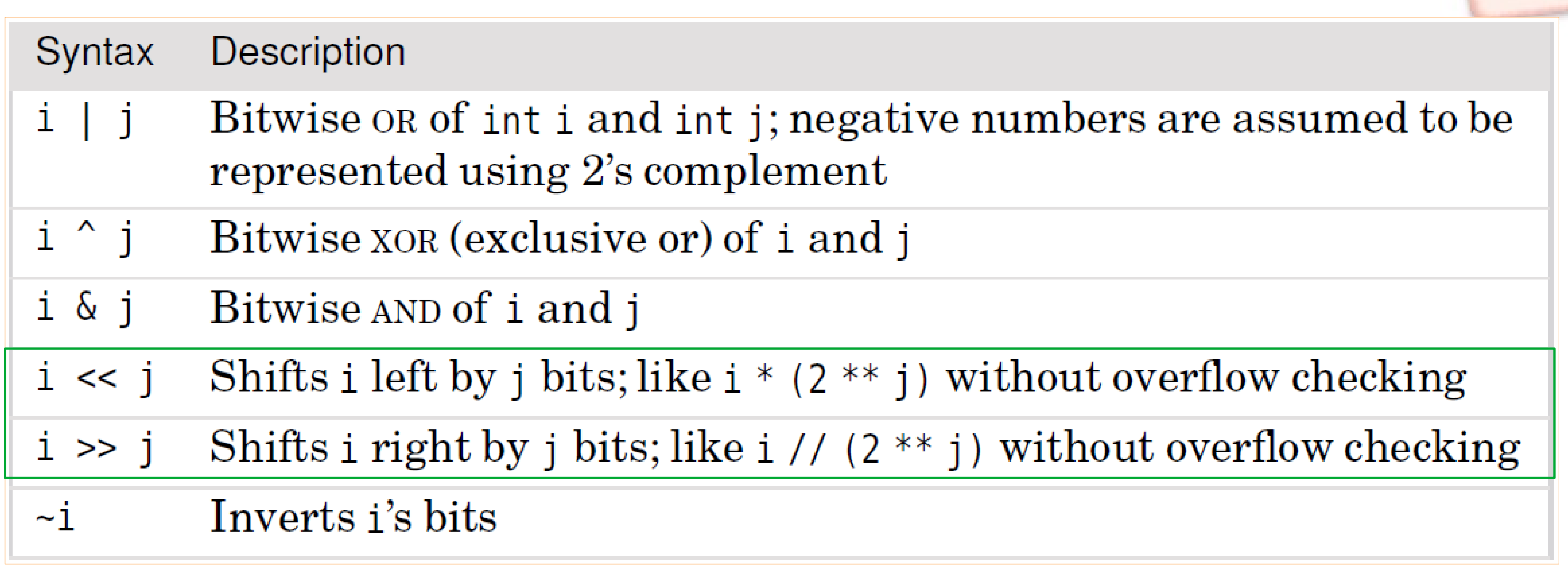


```
z = str(y)
```

```
type(z)
```

What are the types of variable y and z?

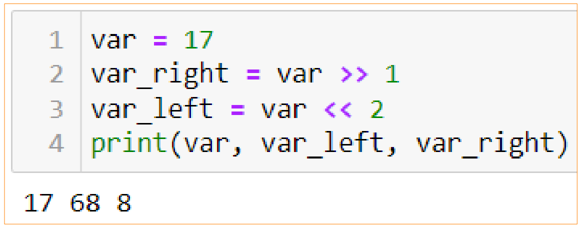
Integer Bit-wise Operators



Syntax	Description
`i	j`	Bitwise OR of `int i` and `int j`; negative numbers are assumed to be represented using 2's complement
`i ^ j`	Bitwise XOR (exclusive or) of `i` and `j`
`i & j`	Bitwise AND of `i` and `j`
`i << j`	Shifts `i` left by `j` bits; like `i * (2 ** j)` without overflow checking
`i >> j`	Shifts `i` right by `j` bits; like `i // (2 ** j)` without overflow checking
`~i`	Inverts `i`'s bits

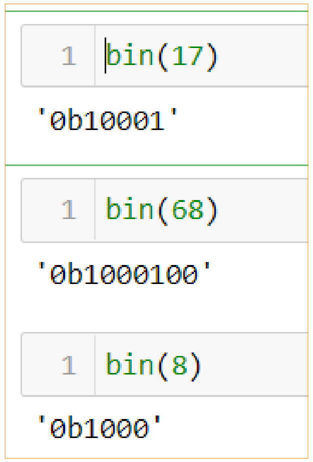
Bit-Shift

- **Binary-left-shift** (**<<**) moves all the digits in the binary number along to the left and fills the gaps after the shift with 0. It is equivalent to a multiplication process for the integer:
  - to multiply by two, all digits shift one place to the left
  - to multiply by four, all digits shift two places to the left
  - and so on
- **Binary-right-shift** (**>>**) moves all the digits in the binary number along to the right and fills the gaps after the shift with 0. It is equivalent to a division process for the integer:
  - to divide by two, all digits shift one place to the right
  - to divide by four, all digits shift two places to the right
  - and so on ...



```
var = 17
var_right = var >> 1
var_left = var << 2
print(var, var_left, var_right)
```

```
17 68 8
```



```
bin(17)
```

```
'0b10001'
```

```
bin(68)
```

```
'0b1000100'
```

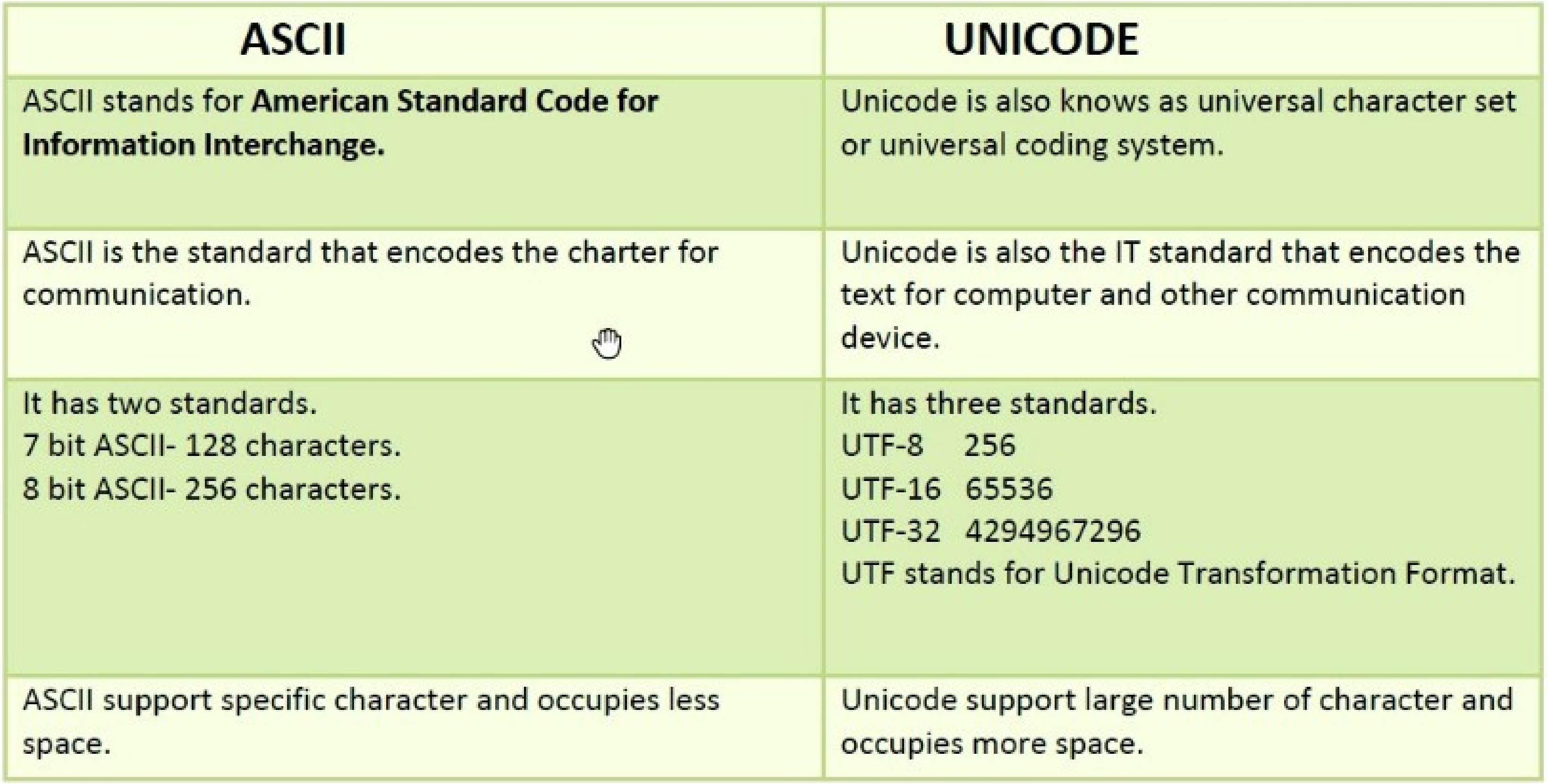
```
bin(8)
```

```
'0b1000'
```

Strings

- String—is an immutable data type which holds a sequence of Unicode characters.
- The Unicode Standard— is an information technology standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems.
  - Unicode can be stored using several different encodings, which translate the character codes into sequences of bytes.
  - The most common encodings are the ASCII-compatible **UTF-8**, the Universal Multiple-Octet Coded Character Set (UCS-2)-compatible **UTF-16**, and **GB18030** which is not an official Unicode standard but is used in China and implements Unicode fully.

Character Encoding



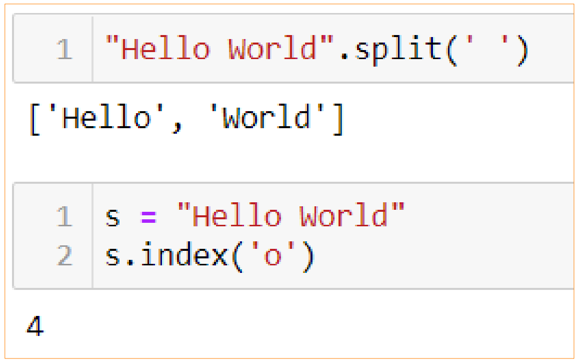
ASCII	UNICODE
ASCII stands for **American Standard Code for Information Interchange.**	Unicode is also knows as universal character set or universal coding system.
ASCII is the standard that encodes the charter for communication.	Unicode is also the IT standard that encodes the text for computer and other communication device.
It has two standards. 7 bit ASCII- 128 characters. 8 bit ASCII- 256 characters.	It has three standards. UTF-8 256 UTF-16 65536 UTF-32 4294967296 UTF stands for Unicode Transformation Format.
ASCII support specific character and occupies less space.	Unicode support large number of character and occupies more space.

String Characteristics

- Strings could be indexed, sliced
- String—is a built-in Python class (OOP) that has several methods.
- To list all methods:

  >> *s = “Hello”*

  >> *s.<press tab button>*



```
"Hello World".split(' ')
```

```
['Hello', 'World']
```

```
s = "Hello World"
s.index('o')
```

```
4
```

String Formatting

Can

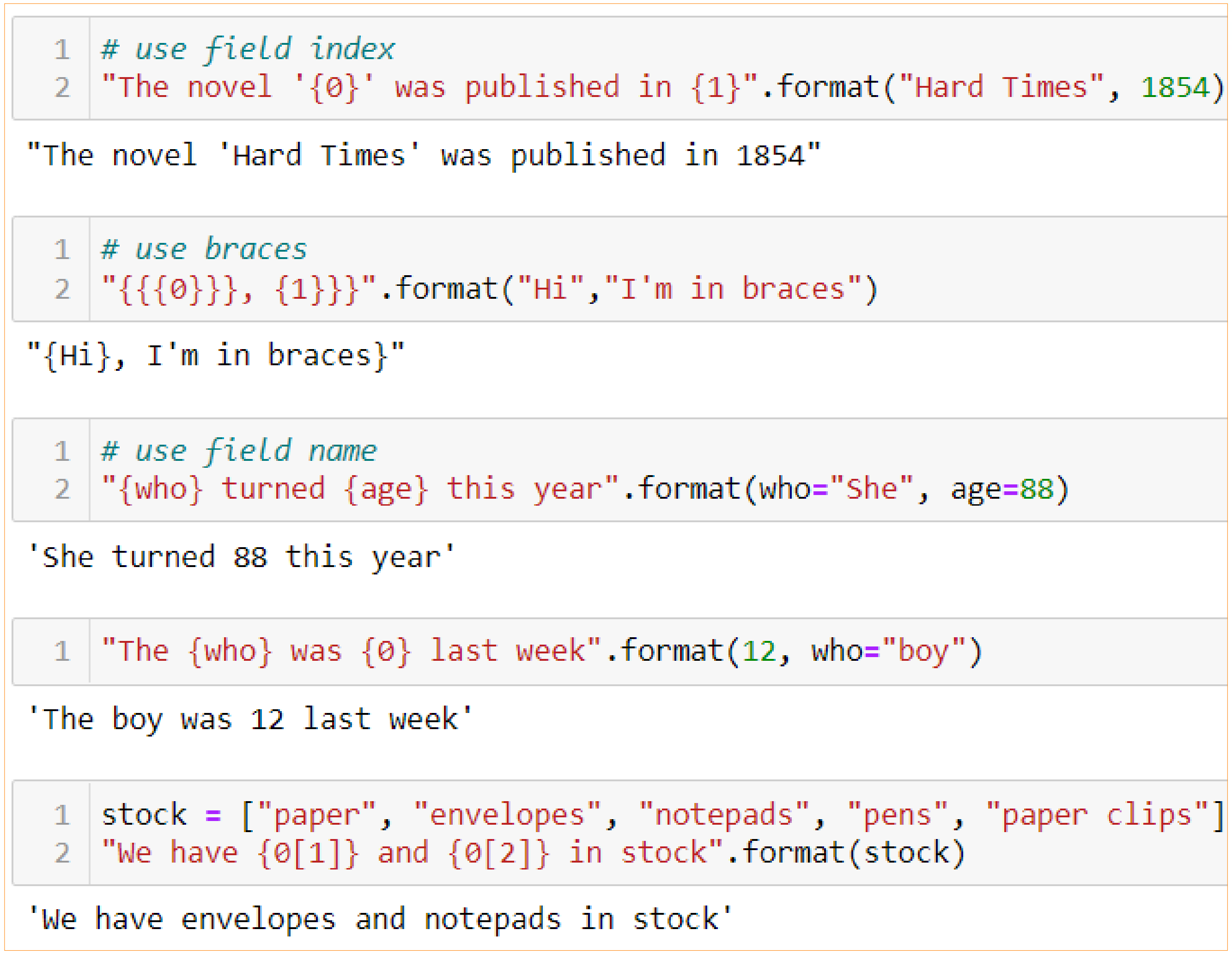
you

detect

how

it

works ...



```python
# use field index
"The novel '{0}' was published in {1}".format("Hard Times", 1854)
```

```
"The novel 'Hard Times' was published in 1854"
```

```python
# use braces
"{{{0}}}, {1}}}".format("Hi","I'm in braces")
```

```
"{Hi}, I'm in braces}"
```

```python
# use field name
"{who} turned {age} this year".format(who="She", age=88)
```

```
'She turned 88 this year'
```

```python
"The {who} was {0} last week".format(12, who="boy")
```

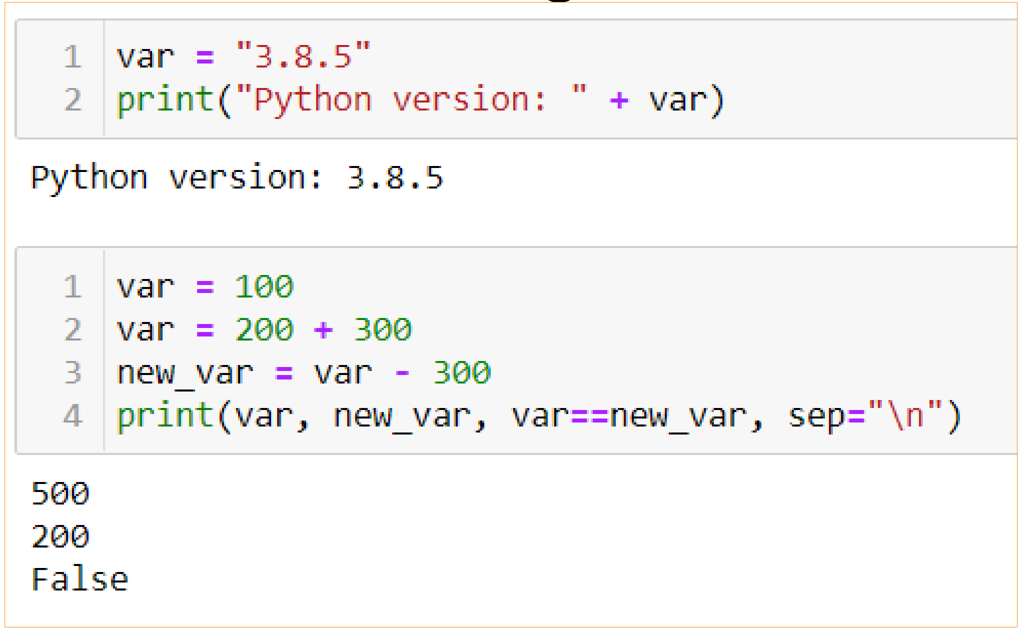
```
'The boy was 12 last week'
```

```python
stock = ["paper", "envelopes", "notepads", "pens", "paper clips"]
"We have {0[1]} and {0[2]} in stock".format(stock)
```

```
'We have envelopes and notepads in stock'
```

Variables

- Variable naming and use



```
var = "3.8.5"
print("Python version: " + var)
```

```
Python version: 3.8.5
```

```
var = 100
var = 200 + 300
new_var = var - 300
print(var, new_var, var==new_var, sep="\n")
```

```
500
200
False
```

  - *Plus operator is used to combine strings...*
  - *What are the legal and illegal variable names?*

Variables and Shortcut Operators

Expression	Shortcut operator
`i = i + 2 * j`	`i += 2 * j`
`var = var / 2`	`var /= 2`
`rem = rem % 10`	`rem %= 10`
`j = j - (i + var + rem)`	`j -= (i + var + rem)`
`x = x ** 2`	`x **= 2`

Quiz

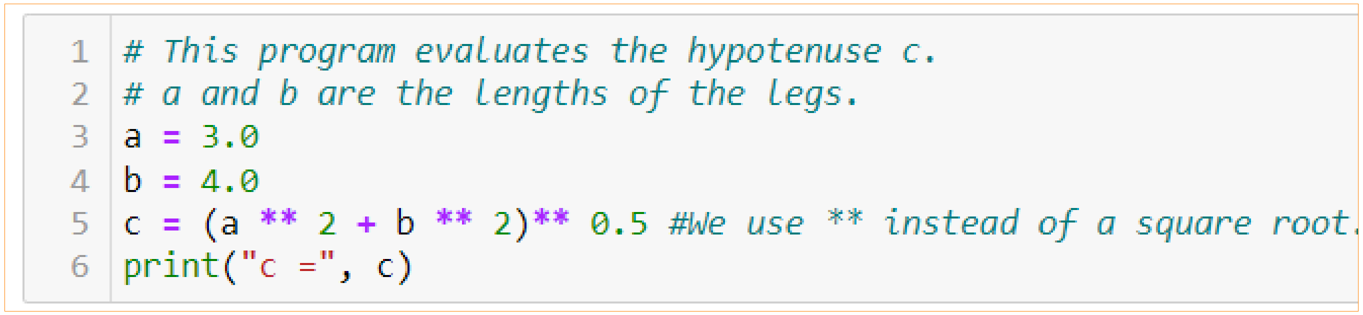
Cisco Networking Academy

Which of the following variable names are illegal in Python? (Select three answers)

- `my_var`
- `m`
- `101`
- `averylongVariablename`
- `m101`
- `m 101`
- `Del`
- `del`

Comments

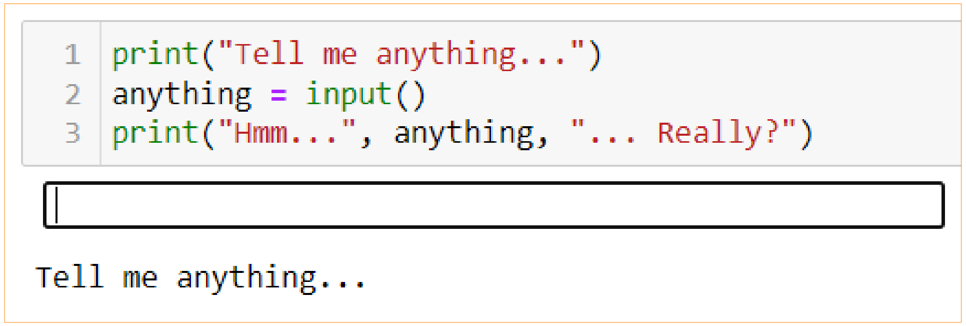
- Why, how and where...
  - Comments operator is **#**
  - Comments could be in a separate line or at the end of a line code
  - You can comment or uncomment a line using keyboard shortcut—l hold ***ctr*** button and press ***/*** button
  - What about commenting multiple lines ? **“””** **...** **“””**



```python
# This program evaluates the hypotenuse c.
# a and b are the lengths of the legs.
a = 3.0
b = 4.0
c = (a ** 2 + b ** 2)** 0.5 #We use ** instead of a square root
print("c =", c)
```

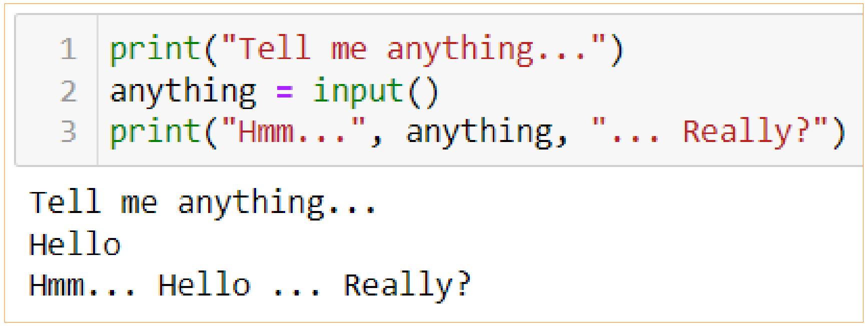
Receive Input from Users

- Interaction with the user—*Input()* function

- 

```
print("Tell me anything...")
anything = input()
print("Hmm...", anything, "... Really?")
```

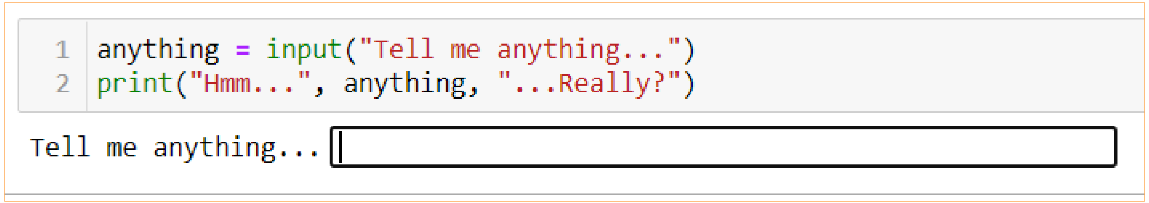
```
Tell me anything...
```

- 

```
print("Tell me anything...")
anything = input()
print("Hmm...", anything, "... Really?")
```

```
Tell me anything...
Hello
Hmm... Hello ... Really?
```

Input() could also print strings

- 

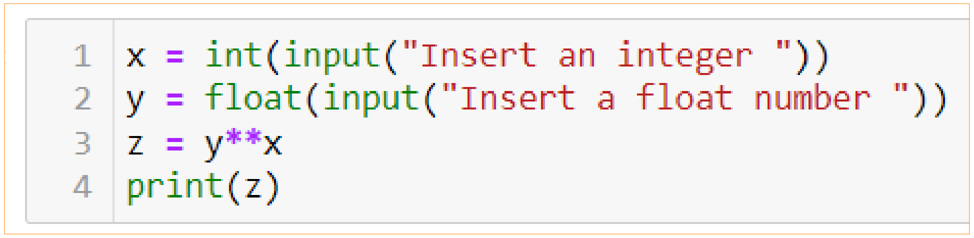
```
anything = input("Tell me anything...")
print("Hmm...", anything, "...Really?")
```

```
Tell me anything...
```

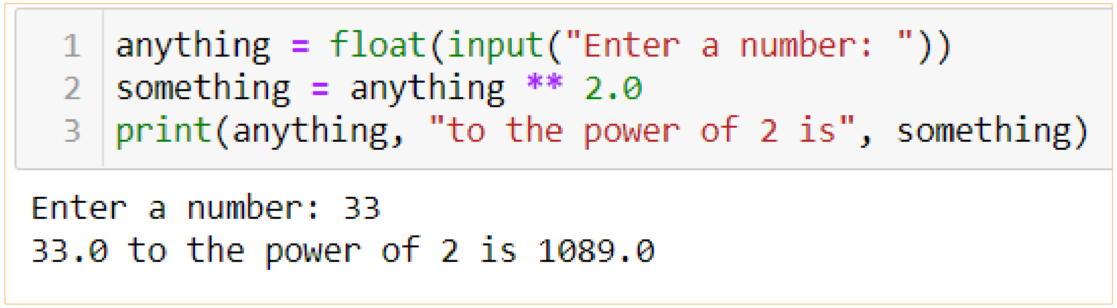
What is the type of the variable anything?

Input() Function

- Use casting to solve that problem

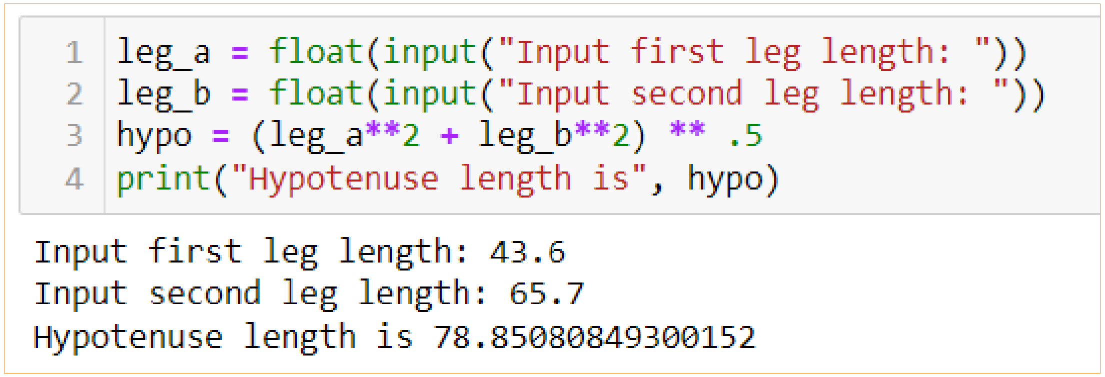


```python
x = int(input("Insert an integer "))
y = float(input("Insert a float number "))
z = y**x
print(z)
```



```python
anything = float(input("Enter a number: "))
something = anything ** 2.0
print(anything, "to the power of 2 is", something)
```

```
Enter a number: 33
33.0 to the power of 2 is 1089.0
```



```python
leg_a = float(input("Input first leg length: "))
leg_b = float(input("Input second leg length: "))
hypo = (leg_a**2 + leg_b**2) ** .5
print("Hypotenuse length is", hypo)
```

```
Input first leg length: 43.6
Input second leg length: 65.7
Hypotenuse length is 78.85080849300152
```

Comparison and Logical Operators

- Comparison Operators

==	***equal***
!=	***not equal***
<	***less than***
>	***greater than***
<=	***less than or equal to***
>=	***greater than or equal to***

- Logical Operators

and, or, not

Programming Task

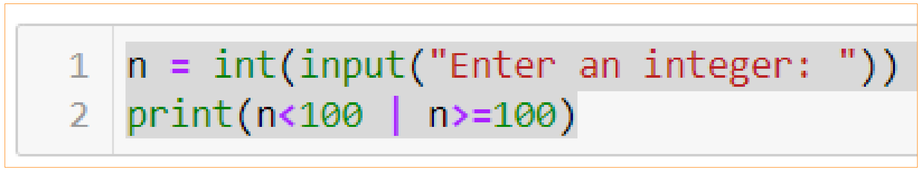
- Using one of the comparison operators in Python, write a simple two-line program that takes the parameter n as input, which is an integer, and prints False if n is less than 100, and True if n is greater than or equal to 100.
- Don’t use the *if* statement…

55	False
99	False
100	True
101	True
-5	False

Programming Task (solution)

- Using one of the comparison operators in Python, write a simple two-line program that takes the parameter n as input, which is an integer, and prints False if n is less than 100, and True if n is greater than or equal to 100.
- Don't use the *if* statement…

55	False
99	False
100	True
101	True
-5	False



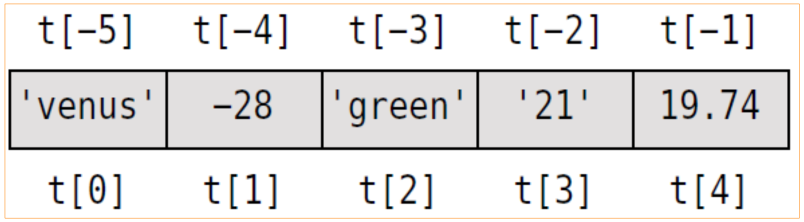
```python
n = int(input("Enter an integer: "))
print(n<100 | n>=100)
```

Collection Data Types

- Collection data types—are data that composed of object elements, such as strings, lists, tuples, sets, and dictionaries.
- Theses data types could be mutable/ immutable, hashable, indexed, ordered, ...
- **Mutable** data—those that could be freely updated at any time. Its items are retrieved in sequence using for or while loops.
- **Hashable** data—are data that is when hashed several times the output remain the same.
- All immutable objects are hashable, but not vice versa.

Tuples and Lists

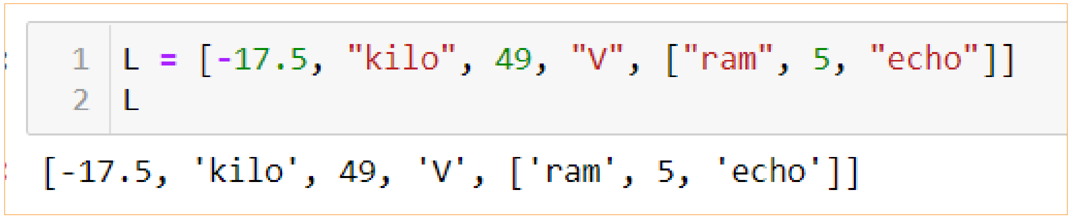
- **Tuple**—it is a sequence of zero or more object references (items are indexed). Tuples are ordered and unchangeable (**immutable**) collections of data. Tuples are written in *round brackets ( )*:



t[-5]	t[-4]	t[-3]	t[-2]	t[-1]
'venus'	-28	'green'	'21'	19.74
t[0]	t[1]	t[2]	t[3]	t[4]

Tuple Index

- **List**—is unordered, changeable (**mutable**), and indexed collection of data. A list is written in *square brackets [ ]*.



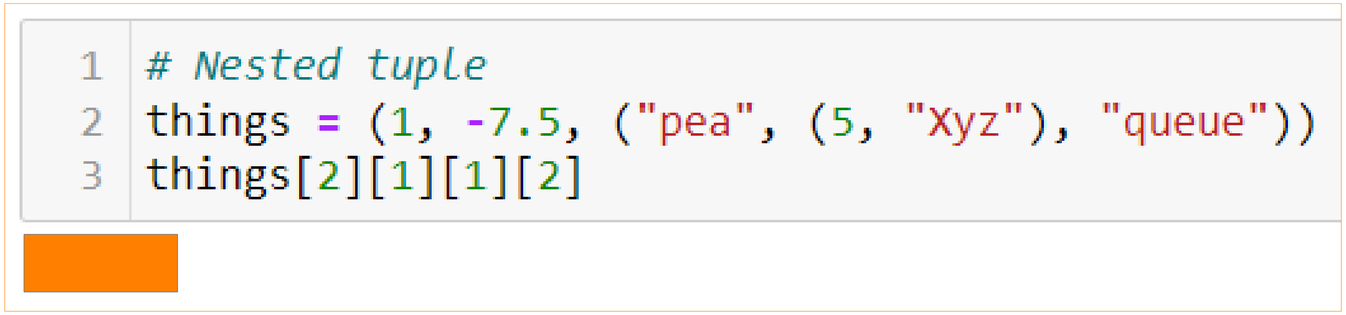
```
L = [-17.5, "kilo", 49, "V", ["ram", 5, "echo"]]
L
```

```
[-17.5, 'kilo', 49, 'V', ['ram', 5, 'echo']]
```

List Index

Nested Tuples and Lists

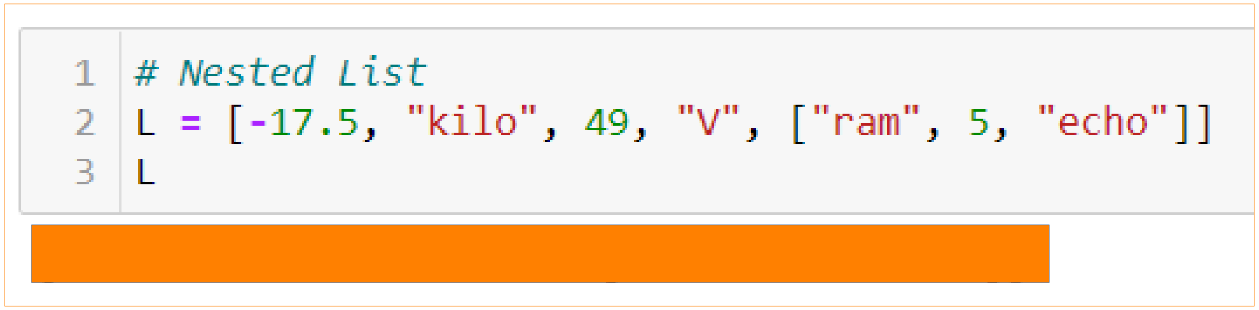
- Tuples and nested tuples



```python
# Nested tuple
things = (1, -7.5, ("pea", (5, "Xyz"), "queue"))
things[2][1][1][2]
```

[redacted]

- Lists and nested lists

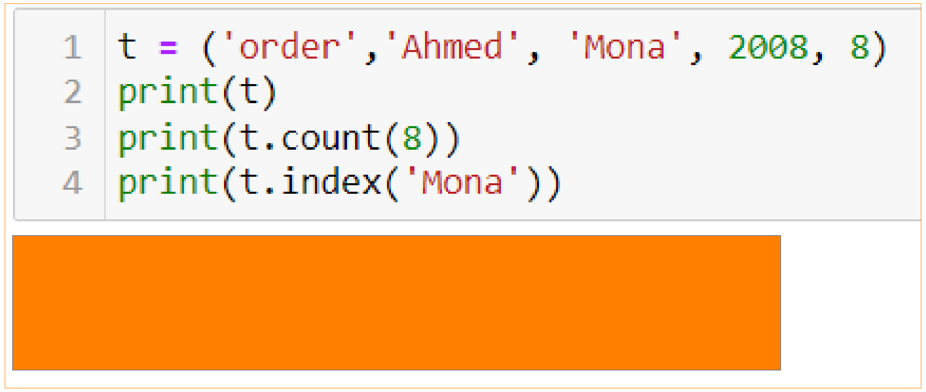


```python
# Nested List
L = [-17.5, "kilo", 49, "V", ["ram", 5, "echo"]]
L
```

[redacted]

Some Examples of Tuple Methods

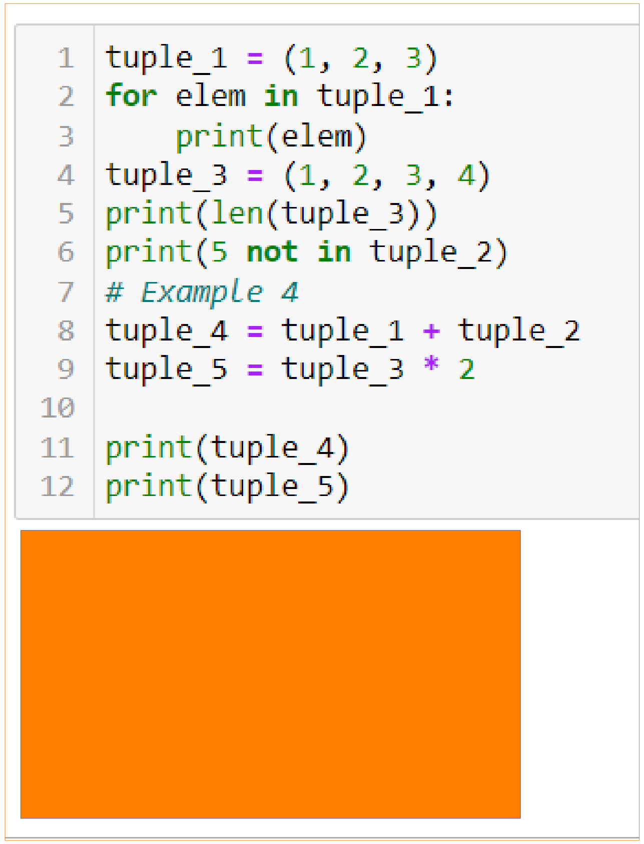
Two tuple methods: count and index



```
t = ('order','Ahmed', 'Mona', 2008, 8)
print(t)
print(t.count(8))
print(t.index('Mona'))
```

[redacted]

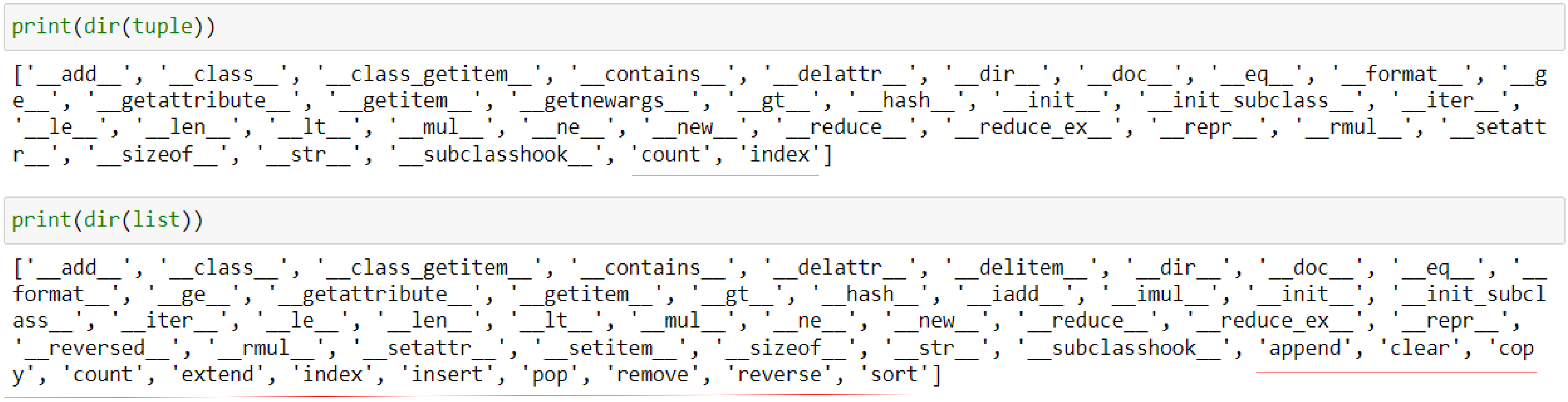
Handling items of a tuple



```
tuple_1 = (1, 2, 3)
for elem in tuple_1:
    print(elem)
tuple_3 = (1, 2, 3, 4)
print(len(tuple_3))
print(5 not in tuple_2)
# Example 4
tuple_4 = tuple_1 + tuple_2
tuple_5 = tuple_3 * 2

print(tuple_4)
print(tuple_5)
```

[redacted]

Tuple and List Methods

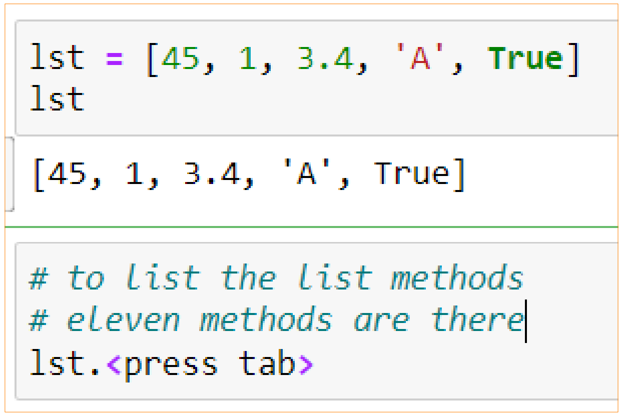


```
print(dir(tuple))
```

```
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__g
e__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__',
'__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmul__', '__setatt
r__', '__sizeof__', '__str__', '__subclasshook__', 'count', 'index']
```

```
print(dir(list))
```

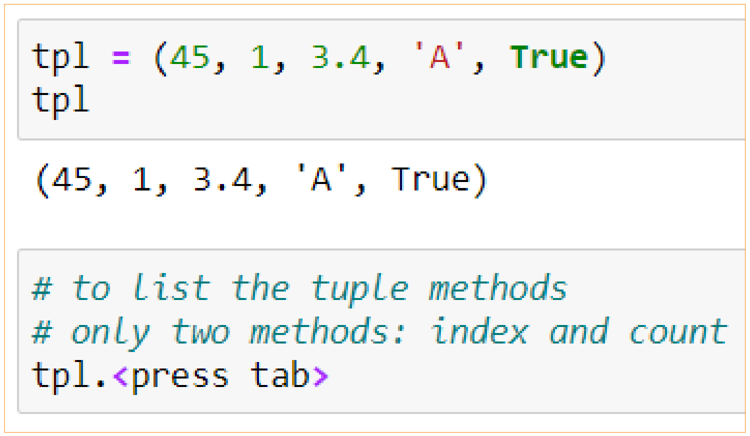
```
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__
format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subcl
ass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'cop
y', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```



```
lst = [45, 1, 3.4, 'A', True]
lst
```

```
[45, 1, 3.4, 'A', True]
```

```
# to list the list methods
# eleven methods are there
lst.<press tab>
```

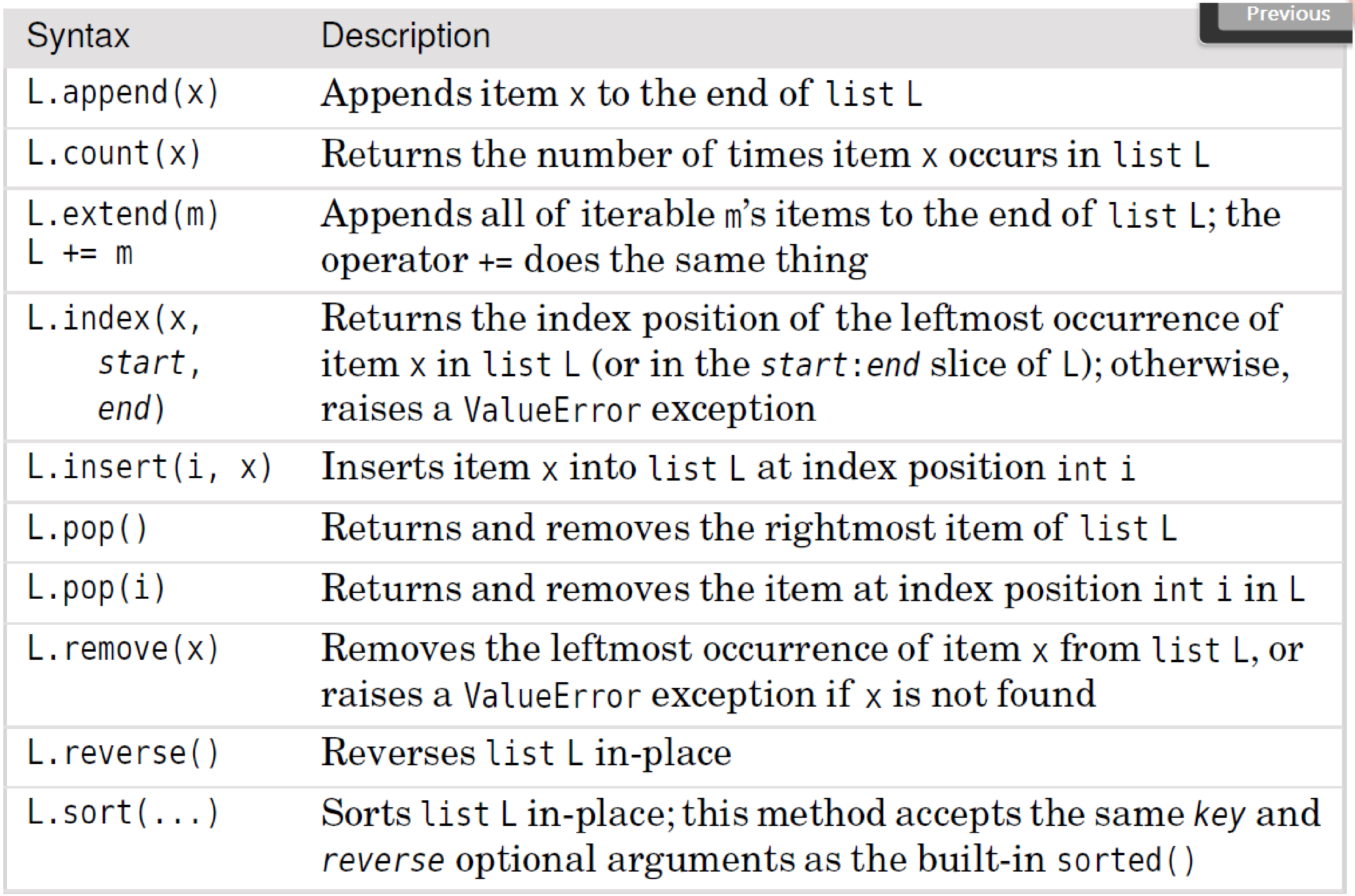


```
tpl = (45, 1, 3.4, 'A', True)
tpl
```

```
(45, 1, 3.4, 'A', True)
```

```
# to list the tuple methods
# only two methods: index and count
tpl.<press tab>
```

List Methods

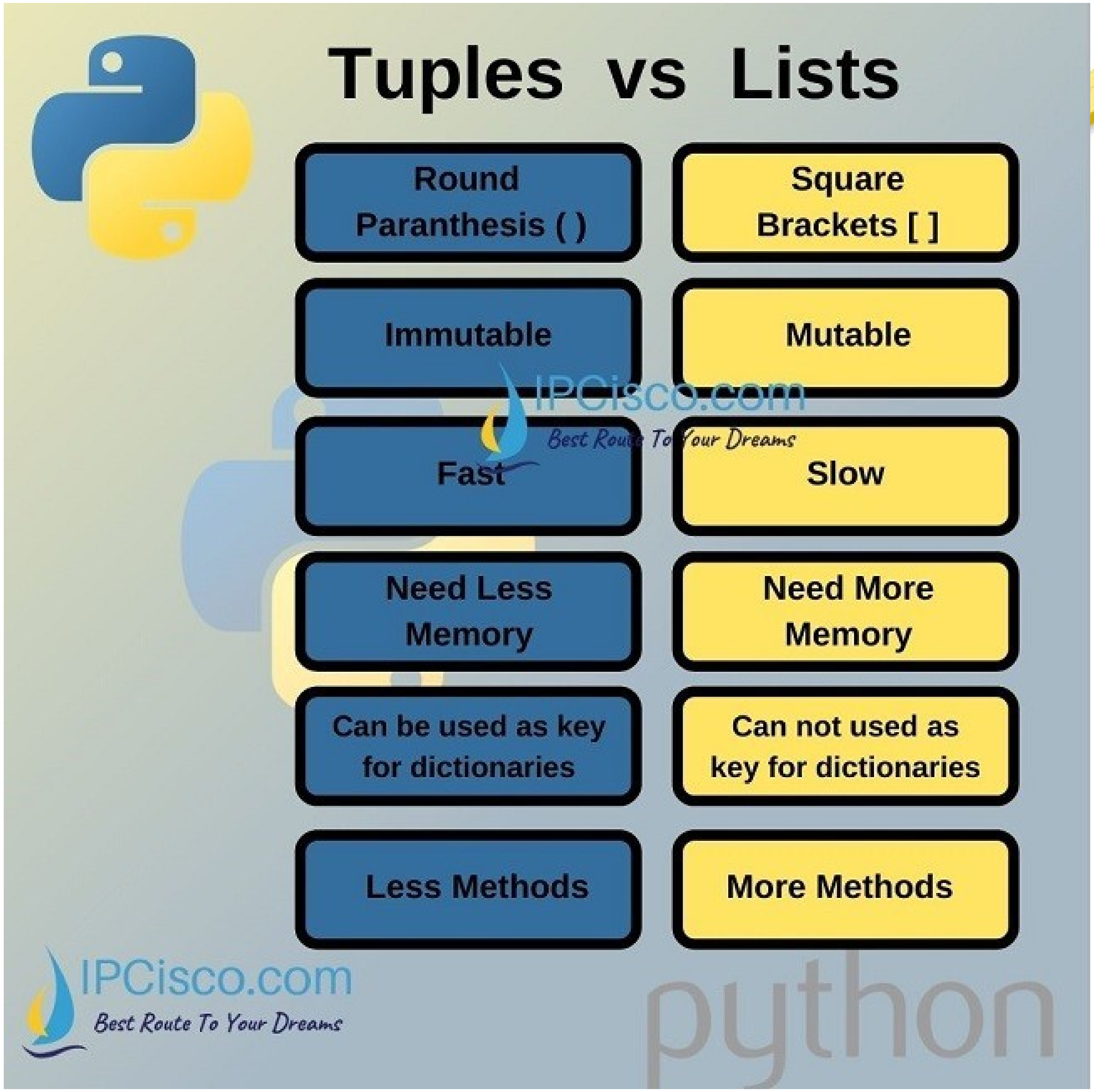


Syntax	Description
`L.append(x)`	Appends item `x` to the end of `list` `L`
`L.count(x)`	Returns the number of times item `x` occurs in `list` `L`

`L.extend(m)`
`L += m`
	Appends all of iterable `m`'s items to the end of `list` `L`; the operator += does the same thing

`L.index(x, start, end)`	Returns the index position of the leftmost occurrence of item `x` in `list` `L` (or in the *start*:*end* slice of `L`); otherwise, raises a `ValueError` exception
`L.insert(i, x)`	Inserts item `x` into `list` `L` at index position `int` `i`
`L.pop()`	Returns and removes the rightmost item of `list` `L`
`L.pop(i)`	Returns and removes the item at index position `int` `i` in `L`
`L.remove(x)`	Removes the leftmost occurrence of item `x` from `list` `L`, or raises a `ValueError` exception if `x` is not found
`L.reverse()`	Reverses `list` `L` in-place
`L.sort(...)`	Sorts `list` `L` in-place; this method accepts the same *key* and *reverse* optional arguments as the built-in `sorted()`

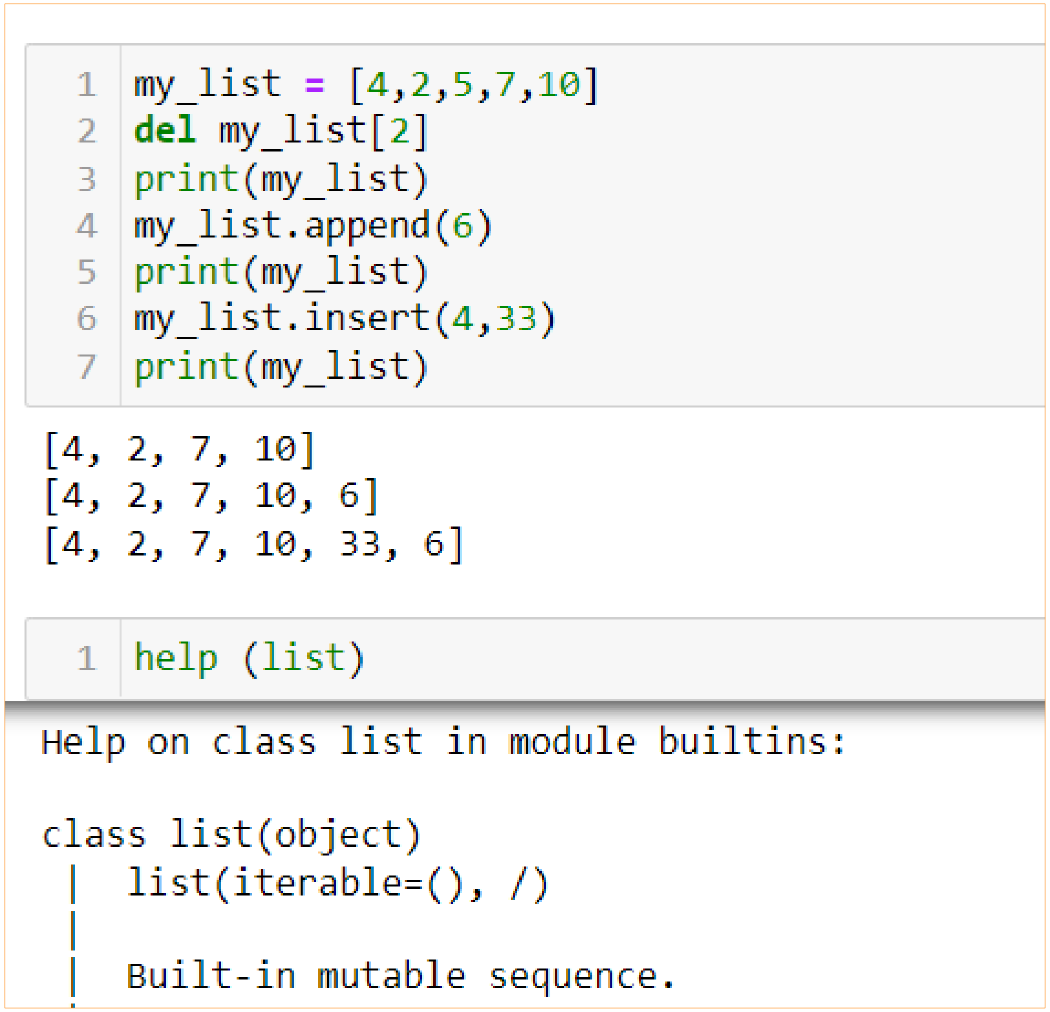
Previous



Tuples vs Lists

Tuples	Lists
Round Paranthesis ( )	Square Brackets [ ]
Immutable	Mutable
Fast	Slow
Need Less Memory	Need More Memory
Can be used as key for dictionaries	Can not used as key for dictionaries
Less Methods	More Methods

Some List Examples

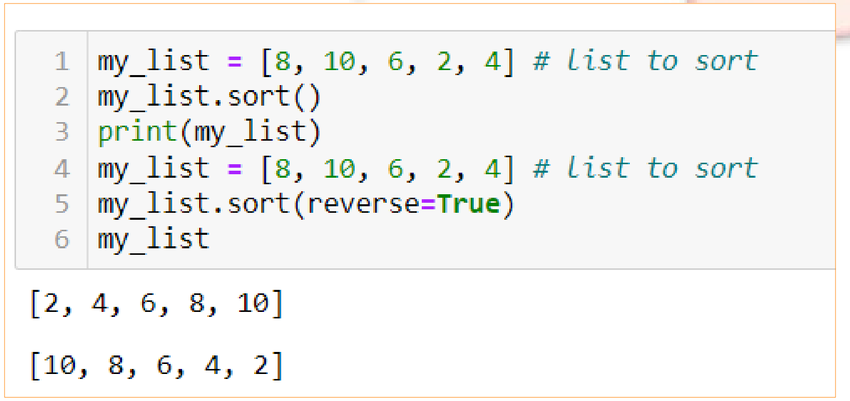


```
my_list = [4,2,5,7,10]
del my_list[2]
print(my_list)
my_list.append(6)
print(my_list)
my_list.insert(4,33)
print(my_list)
```

```
[4, 2, 7, 10]
[4, 2, 7, 10, 6]
[4, 2, 7, 10, 33, 6]
```

```
help (list)
```

```
Help on class list in module builtins:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
```



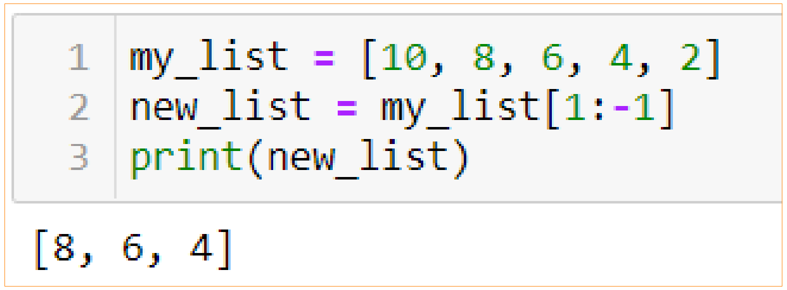
```
my_list = [8, 10, 6, 2, 4] # list to sort
my_list.sort()
print(my_list)
my_list = [8, 10, 6, 2, 4] # list to sort
my_list.sort(reverse=True)
my_list
```

```
[2, 4, 6, 8, 10]

[10, 8, 6, 4, 2]
```

*List.**sort**()*
*list**.append**(value)*
*list**.insert**(location, value)*

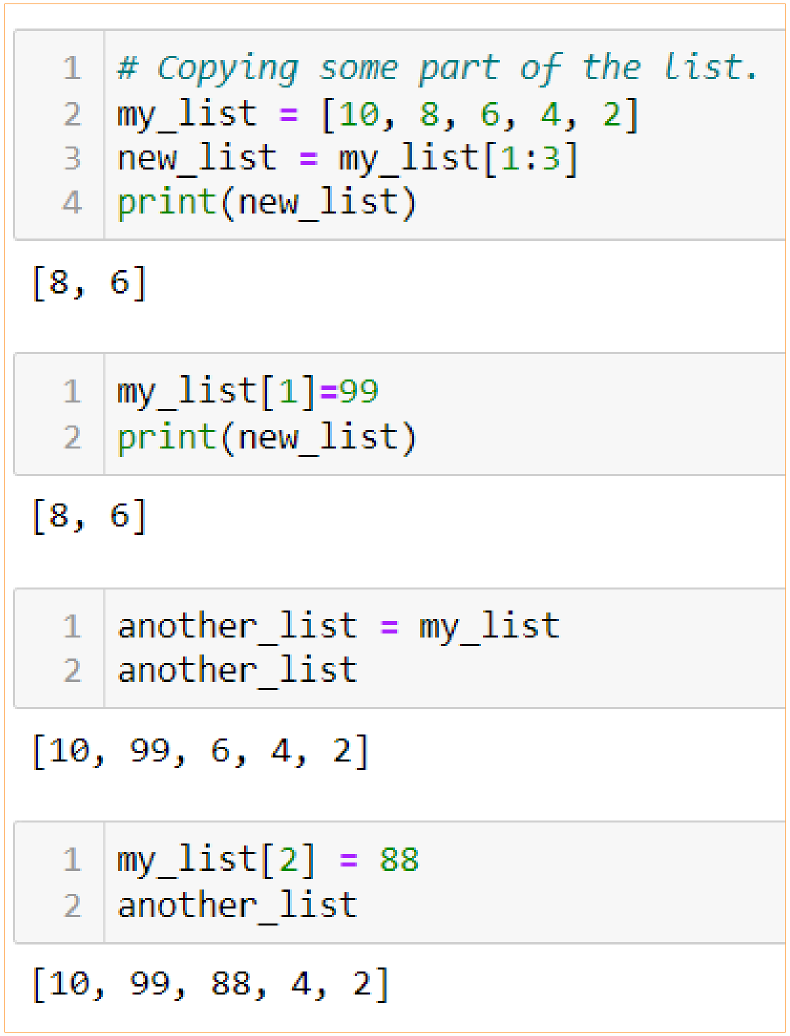
List slicing and indexing

- Slice of a list



```python
my_list = [10, 8, 6, 4, 2]
new_list = my_list[1:-1]
print(new_list)
```

```
[8, 6, 4]
```



```python
# Copying some part of the list.
my_list = [10, 8, 6, 4, 2]
new_list = my_list[1:3]
print(new_list)
```

```
[8, 6]
```

```python
my_list[1]=99
print(new_list)
```

```
[8, 6]
```

```python
another_list = my_list
another_list
```

```
[10, 99, 6, 4, 2]
```

```python
my_list[2] = 88
another_list
```

```
[10, 99, 88, 4, 2]
```

List Comprehension

- it replaces the traditional for loop lines with a single line in the form of:

  [*expression* for *item* in *iterable*]

  [*expression* for *item* in *iterable* if *condition*]

- The first syntax is equivalent to:

```
temp = [ ]
for item in iterable:
    temp.append(expression)
```

- The second syntax is equivalent to:

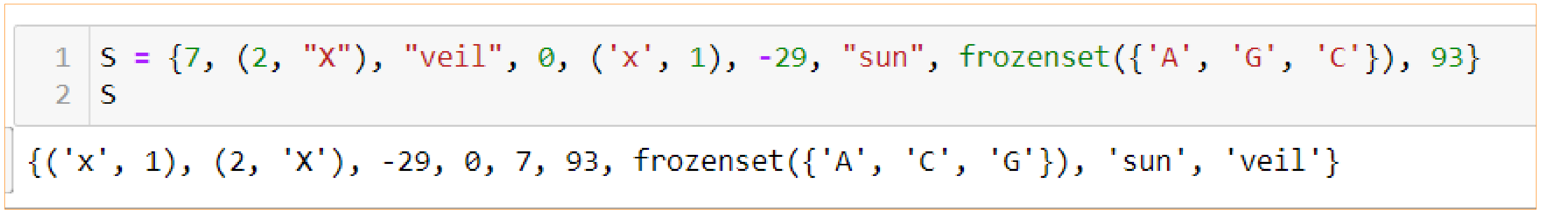
```
temp = [ ]
for item in iterable:
    if condition:
        temp.append(expression)
```

Programming Task

- Write a program that reflects these changes and lets you practice with the concept of lists. Your task is to:
  - step 1: create an empty list named *beatles*;
  - step 2: use the append() method to add the following members of the band to the list: *John Lennon*, *Paul McCartney*, and *George Harrison*;
  - step 3: use the for loop (or list comprehension) and the append() method to prompt the user to add the following members of the band to the list: *Stu Sutcliffe*, and *Pete Best*;
  - step 4: use the del instruction to remove *Stu Sutcliffe* and *Pete Best* from the list;
  - step 5: use the insert() method to add *Ringo Starr* to the beginning of the list.

Sets

- ***Set***—is a set of unordered unique items. Python provides two built-in set types: the mutable set type and the immutable frozenset.
- ***Frozen set***—is a set that, once created, cannot be changed.



```
S = {7, (2, "X"), "veil", 0, ('x', 1), -29, "sun", frozenset({'A', 'G', 'C'}), 93}
S
```

```
{('x', 1), (2, 'X'), -29, 0, 7, 93, frozenset({'A', 'C', 'G'}), 'sun', 'veil'}
```

?

- ***Set comprehension***—is Like list comprehensions. It is defined in which two syntaxes are supported:

  {expression for item in iterable}

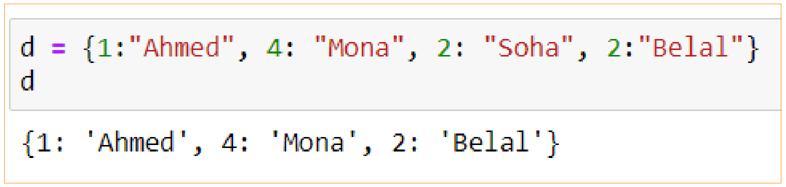
  {expression for item in iterable if condition}

  html = {x for x in files if x.lower().endswith((".htm", ".html"))}

  Given a list of filenames in files, this set comprehension makes the set html hold only those filenames that end in .htm or .html, regardless of case.

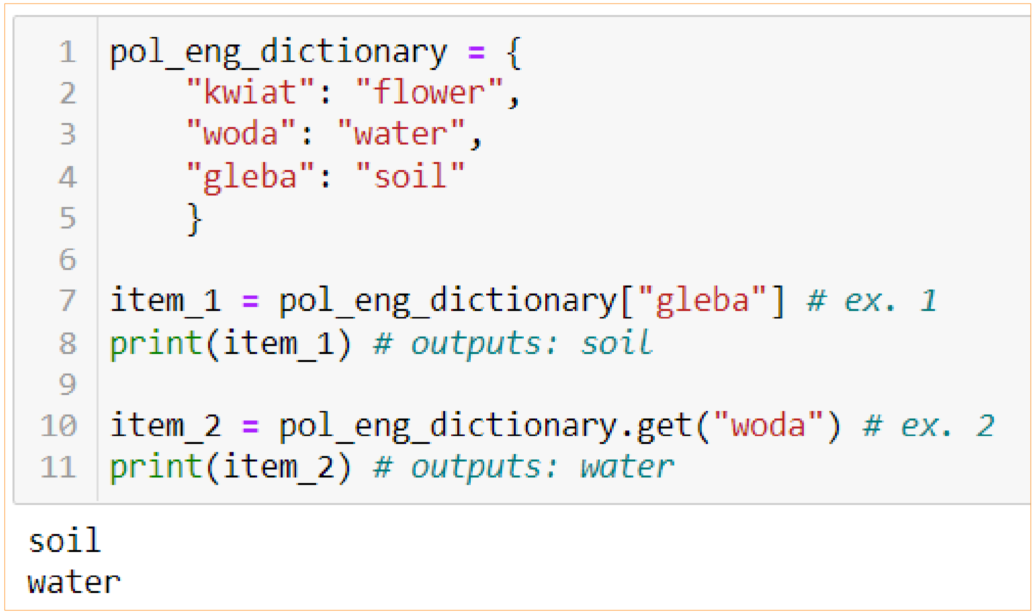
Dictionaries

- Dictionary is an ordered, changeable (**mutable**), and indexed collections of data.
- Each item in a dictionary is composed of key and value



```python
d = {1:"Ahmed", 4: "Mona", 2: "Soha", 2:"Belal"}
d
```

```
{1: 'Ahmed', 4: 'Mona', 2: 'Belal'}
```

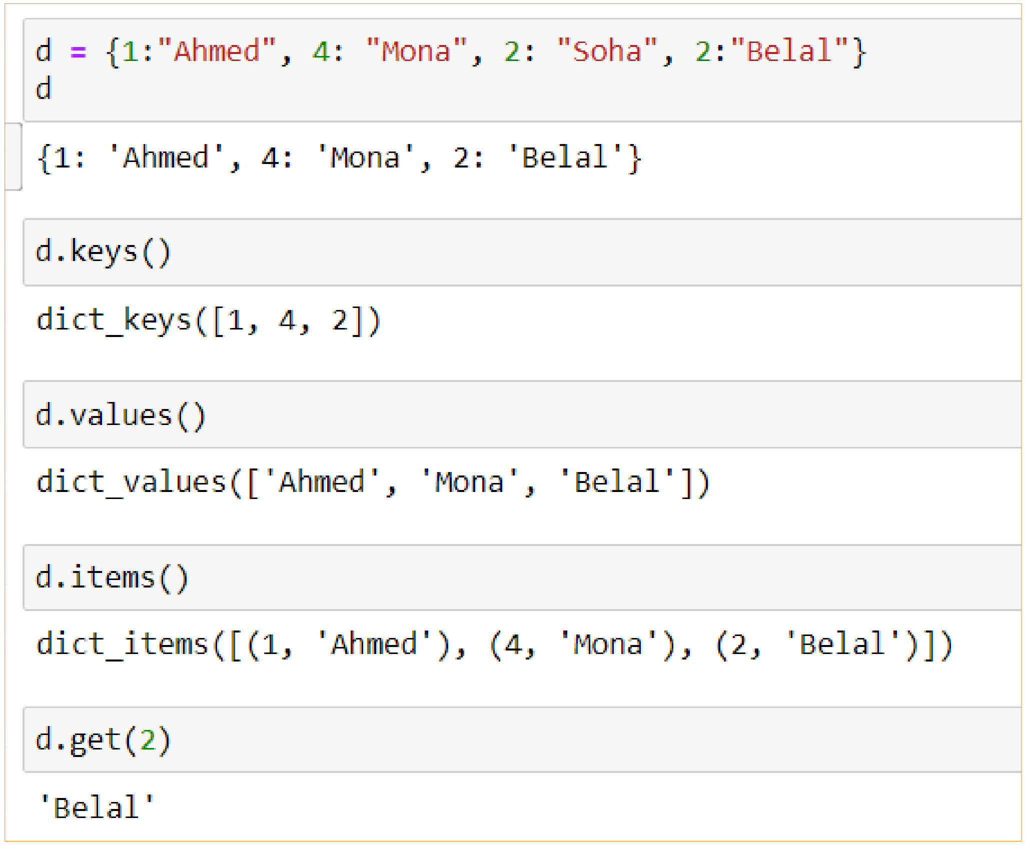


```python
pol_eng_dictionary = {
    "kwiat": "flower",
    "woda": "water",
    "gleba": "soil"
    }

item_1 = pol_eng_dictionary["gleba"] # ex. 1
print(item_1) # outputs: soil

item_2 = pol_eng_dictionary.get("woda") # ex. 2
print(item_2) # outputs: water
```

```
soil
water
```

Some Dictionary Methods

- Keys, values, items, ...



```
d = {1:"Ahmed", 4: "Mona", 2: "Soha", 2:"Belal"}
d
```

```
{1: 'Ahmed', 4: 'Mona', 2: 'Belal'}
```

```
d.keys()
```

```
dict_keys([1, 4, 2])
```

```
d.values()
```

```
dict_values(['Ahmed', 'Mona', 'Belal'])
```

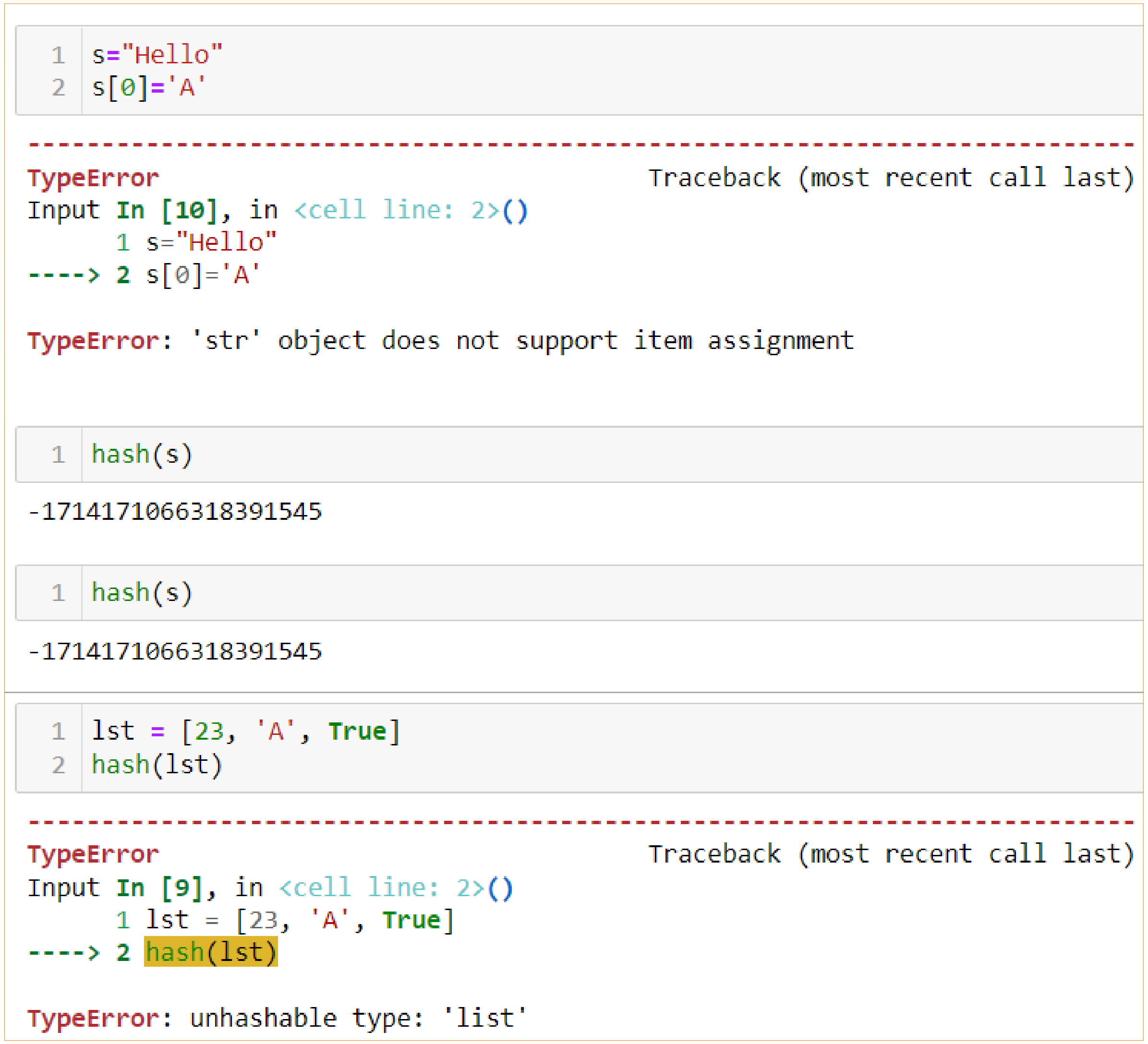
```
d.items()
```

```
dict_items([(1, 'Ahmed'), (4, 'Mona'), (2, 'Belal')])
```

```
d.get(2)
```

```
'Belal'
```

Immutable vs Hashable Objects



```
s="Hello"
s[0]='A'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [10], in <cell line: 2>()
      1 s="Hello"
----> 2 s[0]='A'

TypeError: 'str' object does not support item assignment
```

```
hash(s)
```

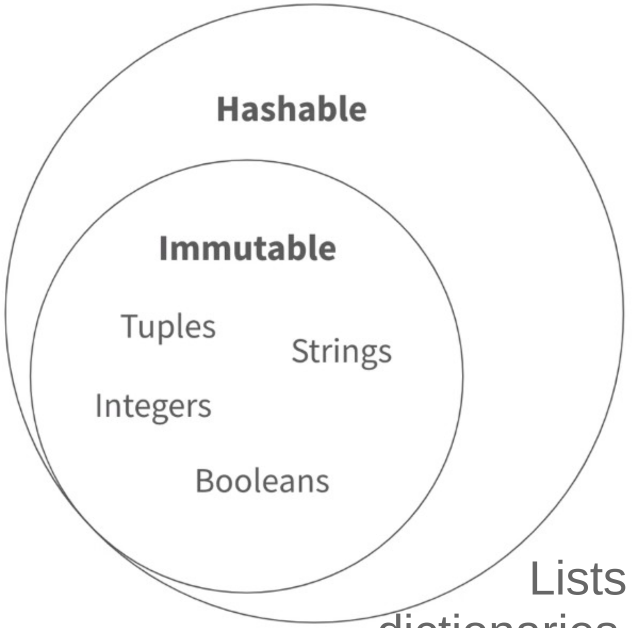
```
-1714171066318391545
```

```
hash(s)
```

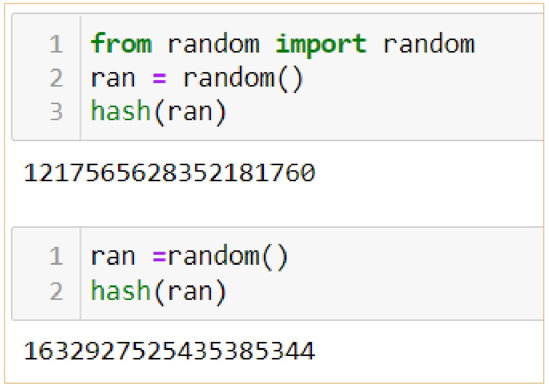
```
-1714171066318391545
```

```
lst = [23, 'A', True]
hash(lst)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [9], in <cell line: 2>()
      1 lst = [23, 'A', True]
----> 2 hash(lst)

TypeError: unhashable type: 'list'
```



Lists
dictionaries



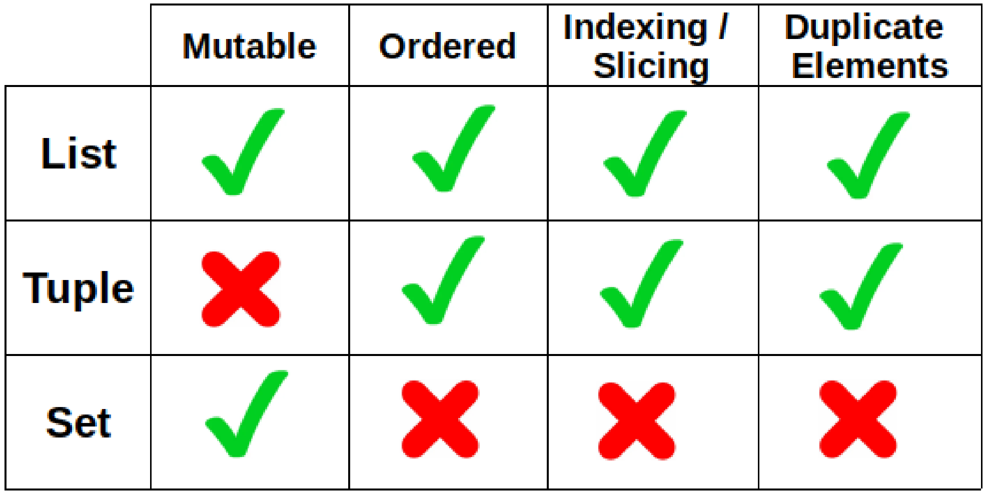
```
from random import random
ran = random()
hash(ran)
```

```
1217565628352181760
```

```
ran =random()
hash(ran)
```

```
1632927525435385344
```

Compare lists, tuples, sets and Dictionaries



	Mutable	Ordered	Indexing / Slicing	Duplicate Elements	
List	✓	✓	✓	✓	[item1, item2, …]
Tuple	✗	✓	✓	✓	(item1, item2, …)
Set	✓	✗	✗	✗	{item1, item2, ...}
Dictionary	✓	✓	✓	✗	{key:value, …}

Conditional Statements

- ***If*** Statements

```
if true_or_false_condition:
    perform_if_condition_true
else:
    perform_if_condition_false
```

- Nested ***if*** and **elif** statements

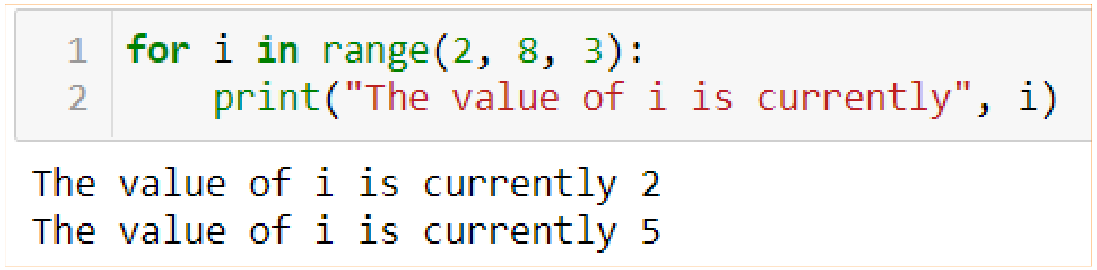
```
if the_weather_is_good:
    if nice_restaurant_is_found:
        have_lunch()
    else:
        eat_a_sandwich()
else:
    if tickets_are_available:
        go_to_the_theater()
    else:
        go_shopping()
```

```
if the_weather_is_good:
    go_for_a_walk()
elif tickets_are_available:
    go_to_the_theater()
elif table_is_available:
    go_for_lunch()
else:
    play_chess_at_home()
```

Control Statements (Loops)

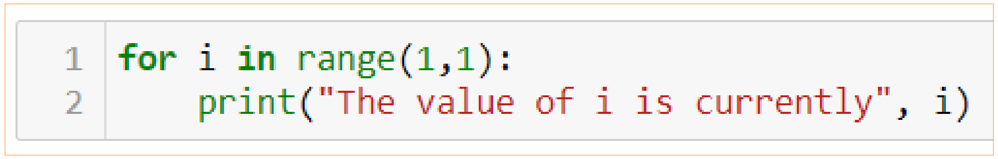
- Why and how?

  for, ***while***, ***break***, ***continue***



```python
for i in range(2, 8, 3):
    print("The value of i is currently", i)
```

```
The value of i is currently 2
The value of i is currently 5
```



```python
for i in range(1,1):
    print("The value of i is currently", i)
```

Note: the set generated by the range() has to be sorted in ascending order. There's no way to force the range() to create a set in a different form when the range() function accepts exactly two arguments. This means that the range()'s second argument must be greater than the first.

Thus, there will be no output here, either:

Methods and Functions

- A typical function invocation may look like this:

  result = function(arg)

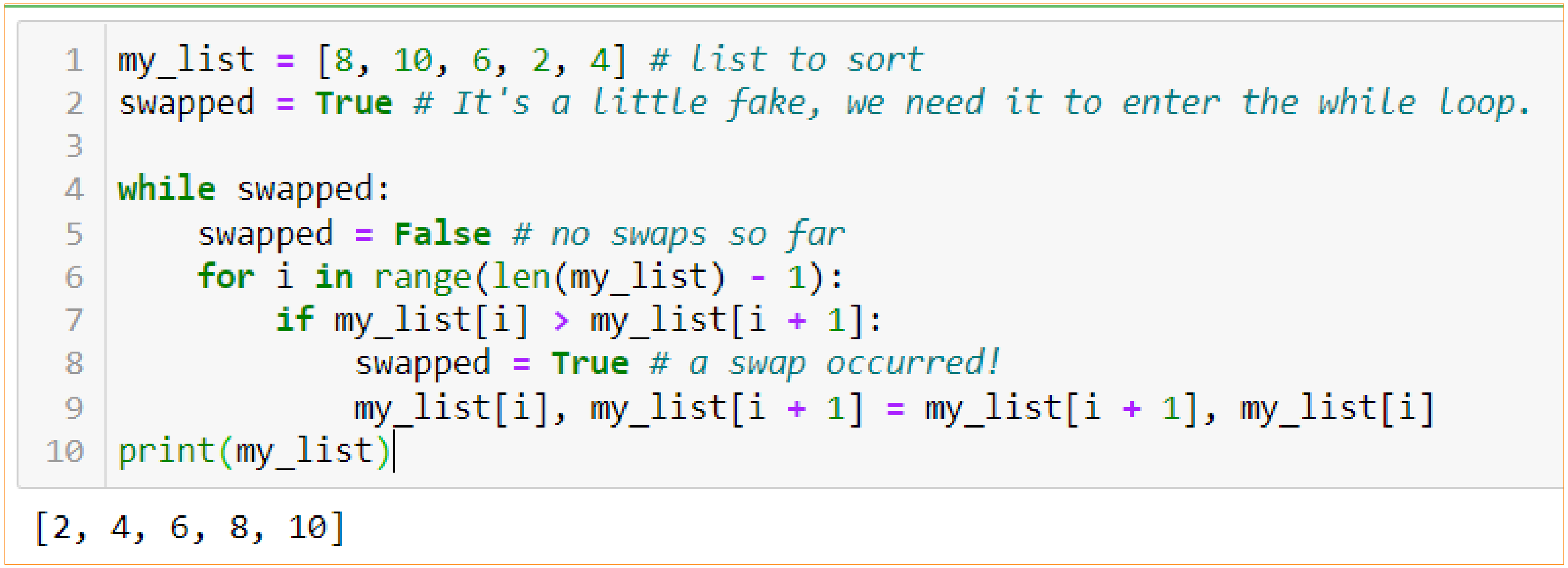
- A typical method invocation may look like this:

  result = data.method(arg)

- A method is owned by the data it works for, while a function is owned by the whole code.
- Both may have argument(s) and return value(s)

The Bubble Sort Algorithm

- Compare subsequent elements and do swapping if needed.

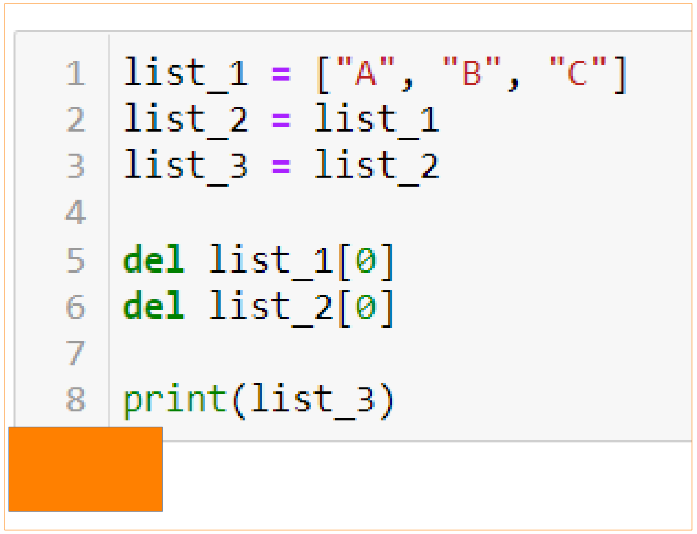


```python
my_list = [8, 10, 6, 2, 4] # list to sort
swapped = True # It's a little fake, we need it to enter the while loop.

while swapped:
    swapped = False # no swaps so far
    for i in range(len(my_list) - 1):
        if my_list[i] > my_list[i + 1]:
            swapped = True # a swap occurred!
            my_list[i], my_list[i + 1] = my_list[i + 1], my_list[i]
print(my_list)
```

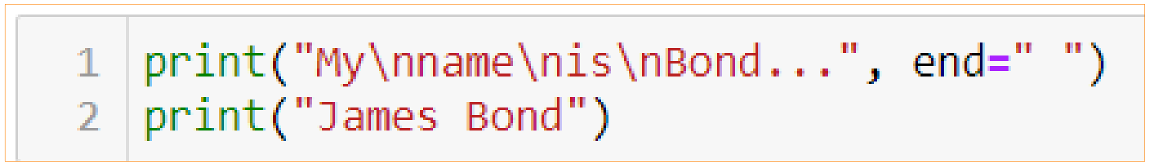
```
[2, 4, 6, 8, 10]
```

Quiz

- What is the output of these codes …



```python
list_1 = ["A", "B", "C"]
list_2 = list_1
list_3 = list_2

del list_1[0]
del list_2[0]

print(list_3)
```



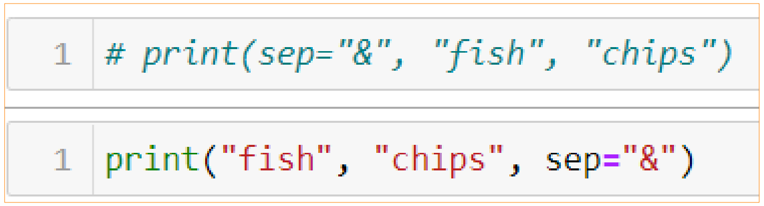
```python
print("My\nname\nis\nBond...", end=" ")
print("James Bond")
```

Programming Tasks

- Write a code to get the following output

  Programming***Essentials***in Python…

- Examine the following statements, separately

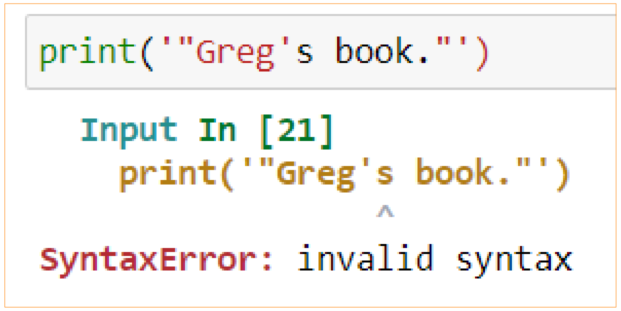


```
# print(sep="&", "fish", "chips")
```

```
print("fish", "chips", sep="&")
```

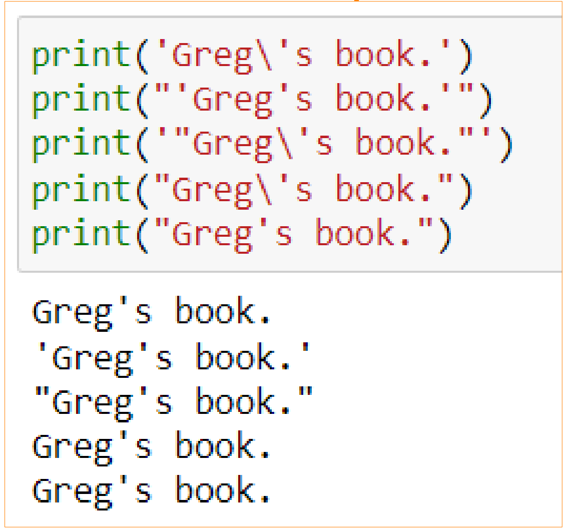
Remember: Keyword arguments should be passed after any required positional arguments.

- Printing special characters



```
print('"Greg's book."')
```

```
  Input In [21]
    print('"Greg's book."')
                ^
SyntaxError: invalid syntax
```



```
print('Greg\'s book.')
print("'Greg's book.'")
print('"Greg\'s book."')
print("Greg\'s book.")
print("Greg's book.")
```

```
Greg's book.
'Greg's book.'
"Greg's book."
Greg's book.
Greg's book.
```

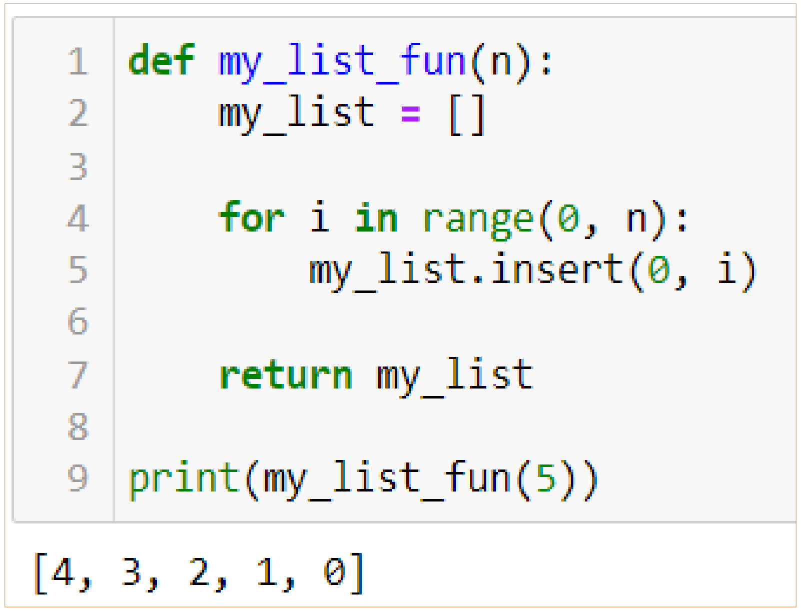
Creating Functions

- For debugging/reviewing/modifying your code, it is better to divide it into small pieces...
- **Function**—It is the piece of code that could be repeated many times from different places in your program.
- A function may have argument(s) or/and return value(s)
- Types of functions: built-in or user defined
- Built-in Function example: print(“Hello World”) … print is a function, its argument is the string “Hello World”, it doesn’t have return value.
- User-defined Function:

```
def function_name():
    function_body
```

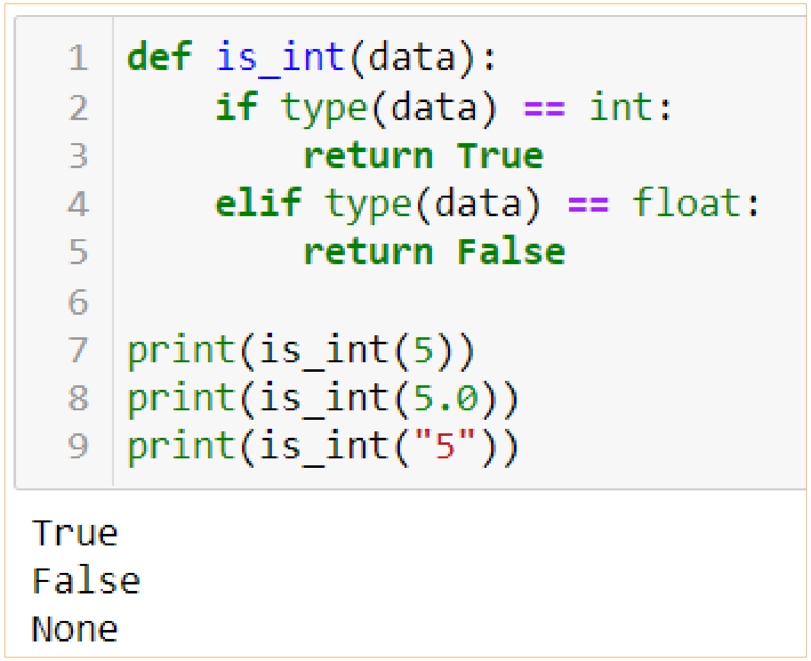
Functions and Name Scope

- User defined function examples



```python
def my_list_fun(n):
    my_list = []

    for i in range(0, n):
        my_list.insert(0, i)

    return my_list

print(my_list_fun(5))
```

```
[4, 3, 2, 1, 0]
```



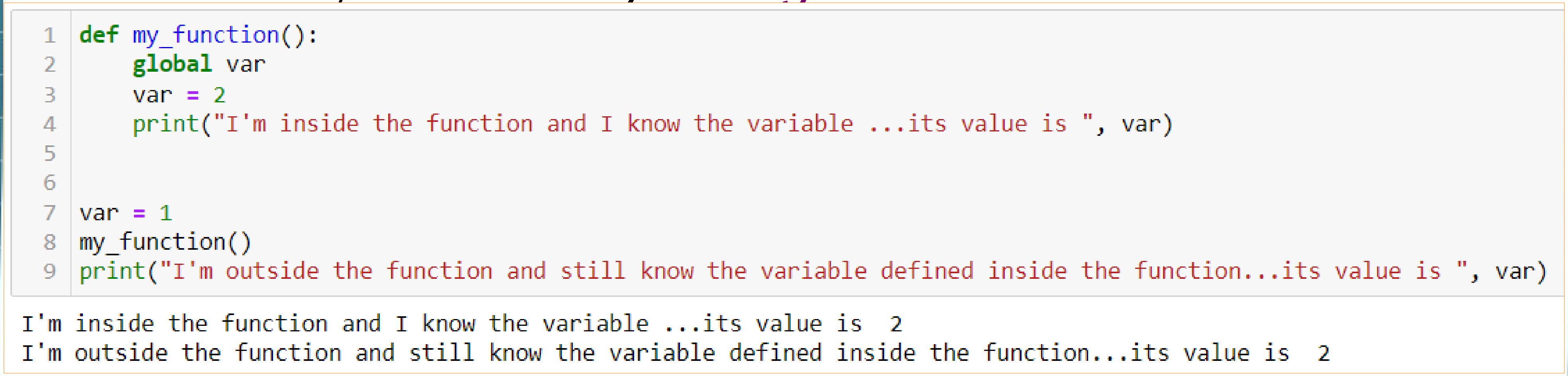
```python
def is_int(data):
    if type(data) == int:
        return True
    elif type(data) == float:
        return False

print(is_int(5))
print(is_int(5.0))
print(is_int("5"))
```

```
True
False
None
```

- Scope of names (e.g. variable names)—if you defined a variable inside a function, it is not known outside it (such as *my_list)*

Functions and Name Scope (cont.)

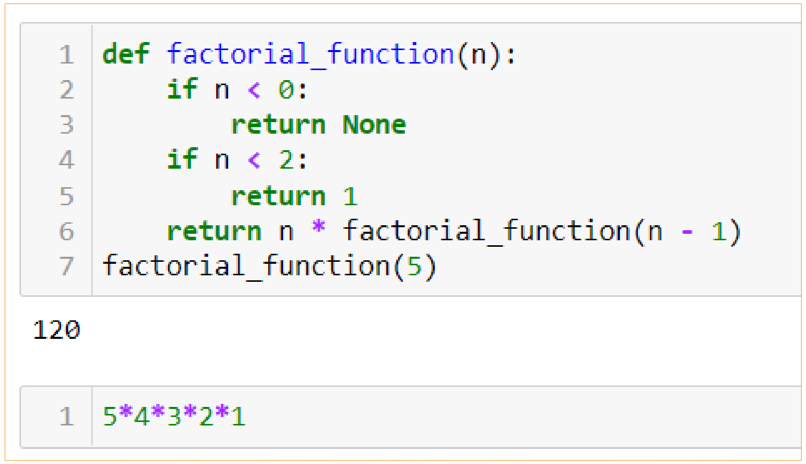
- If you want to extend the scope of a variable defined inside a function, use the keyword *global*



```python
def my_function():
    global var
    var = 2
    print("I'm inside the function and I know the variable ...its value is ", var)


var = 1
my_function()
print("I'm outside the function and still know the variable defined inside the function...its value is ", var)
```

```
I'm inside the function and I know the variable ...its value is  2
I'm outside the function and still know the variable defined inside the function...its value is  2
```

- Recursion—is the function call to itself...



```python
def factorial_function(n):
    if n < 0:
        return None
    if n < 2:
        return 1
    return n * factorial_function(n - 1)
factorial_function(5)
```

```
120
```

```python
5*4*3*2*1
```

“RecursionError: maximum recursion depth exceeded” … will be raised if you delete the < 0 and > 2 check

Handling Errors—Try … Exceptions

- Basic Exception Handling:

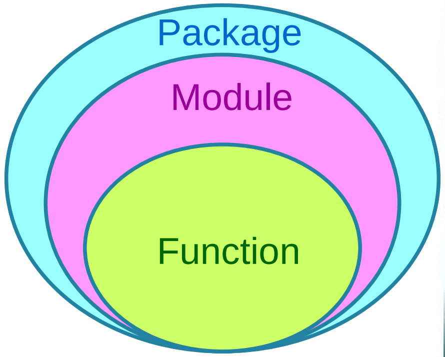
```
try:
    – try_suite
except exception1 as variable1:
    – exception_suite1
…
except exceptionN as variableN:
    – exception_suiteN
```

Packages, Modules, and PIP

- Modules—are identified by its name. If you want to use any module, you need to know its name.
  - Python has a large number of modules.
  - All these modules, along with the built-in functions, form the Python standard library
  - A full list of all "volumes" in that library is found at: https://docs.python.org/3/library/index.html.
  - Each module consists of entities. These entities can be functions, variables, constants, classes, and objects.
  - To use an entity of a module, you should import the module

Packages

- A Python package can contains several modules. Simply, a package is a folder that contains various modules as files.
- You could create your own package(s), module(s), and function(s)...*envoke a Python shell to practice this practice.*
  - Creating a package:
    - Create a folder named as the package name.
    - A package directory should have a file named __init__.py, it is used for initializing the package and may be empty, but not absent.
    - Inside this folder create Python files as the name of your modules
    - Inside each module file you can create your functions

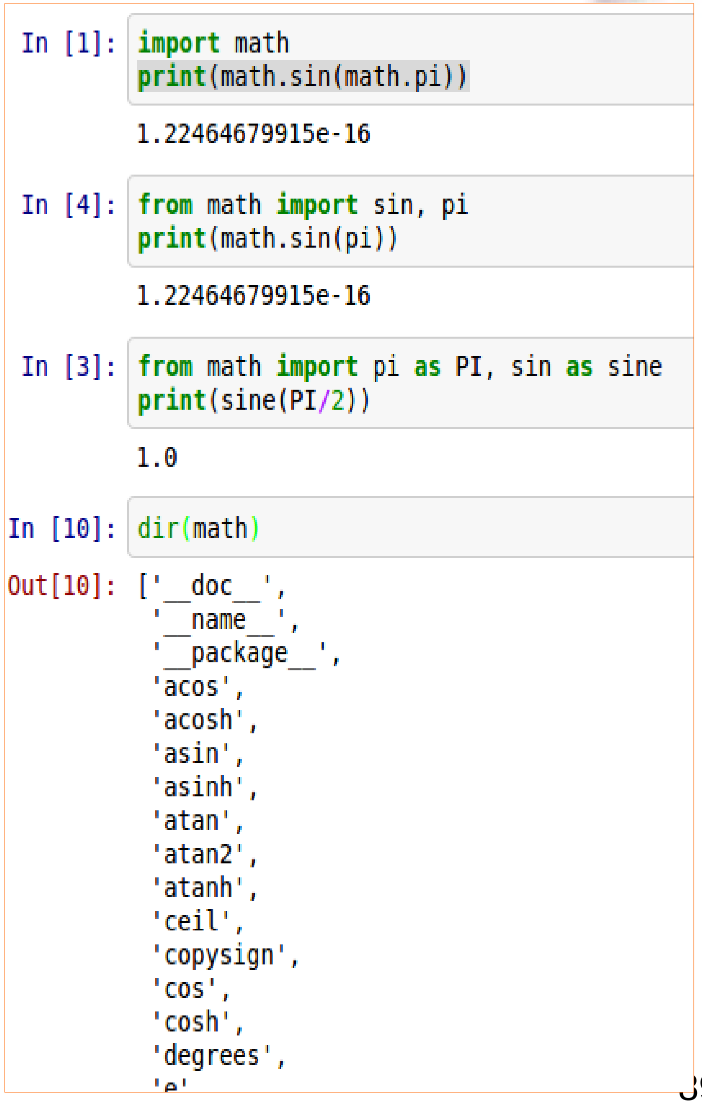


Modules

- Importing and using modules—using *import* or *from..import* keywords
  - from <moduleName> import * is the same as import <moduleName>, except that you don't need to put the module name before the method name...
  - import <moduleName> as <aliasName>
  - To list all methods of a module

    dir(<moduleName>)

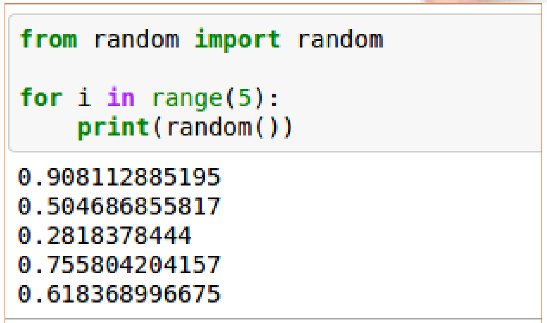
Task: could display the output of dir(math) with tab separator instead of newline



```
In [1]: import math
        print(math.sin(math.pi))

        1.22464679915e-16

In [4]: from math import sin, pi
        print(math.sin(pi))

        1.22464679915e-16

In [3]: from math import pi as PI, sin as sine
        print(sine(PI/2))

        1.0

In [10]: dir(math)

Out[10]: ['__doc__',
          '__name__',
          '__package__',
          'acos',
          'acosh',
          'asin',
          'asinh',
          'atan',
          'atan2',
          'atanh',
          'ceil',
          'copysign',
          'cos',
          'cosh',
          'degrees',
          'e'
```

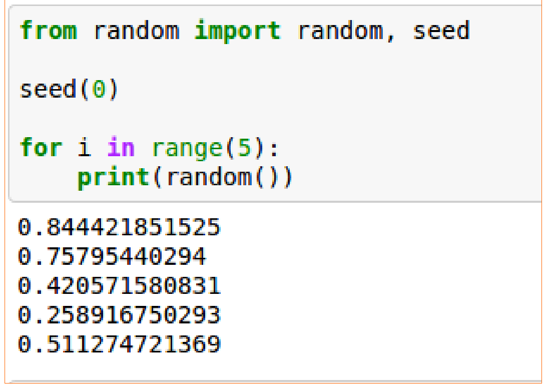
Modules (cont.)

- Random Module
  - random, seed, randint, and randrange functions
    - *random*—produces a float number x coming from the range (0.0, 1.0).
    - *seed(int_value)*—sets the seed with the integer value int_value. This wil produce the same random numbers every time you run your code.
    - *seed( )*—sets the seed with the current time.
    - *randrange(beg, end, step)*—determines the range of the random number.
    - Use *choice* and *sample* functions to select random number(s) from a specified list



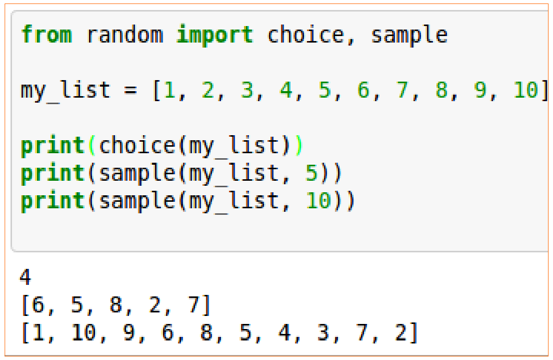
```
from random import random

for i in range(5):
    print(random())
```

```
0.908112885195
0.504686855817
0.2818378444
0.755804204157
0.618368996675
```



```
from random import random, seed

seed(0)

for i in range(5):
    print(random())
```

```
0.844421851525
0.75795440294
0.420571580831
0.258916750293
0.511274721369
```

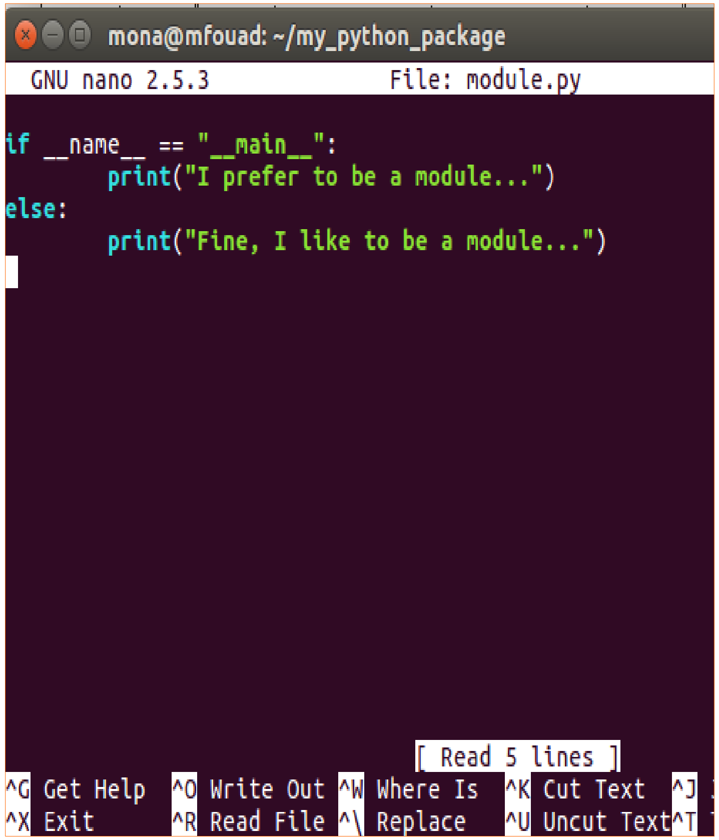


```
from random import choice, sample

my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(choice(my_list))
print(sample(my_list, 5))
print(sample(my_list, 10))
```

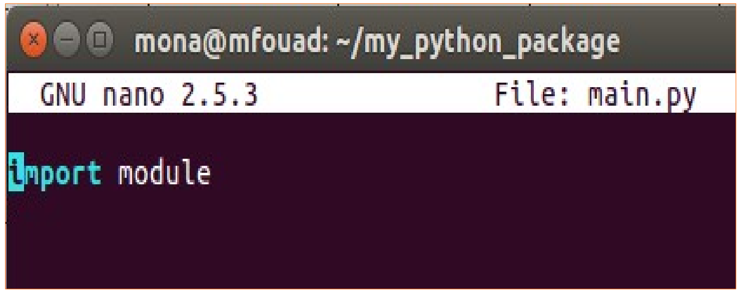
```
4
[6, 5, 8, 2, 7]
[1, 10, 9, 6, 8, 5, 4, 3, 7, 2]
```

Modules (cont.)

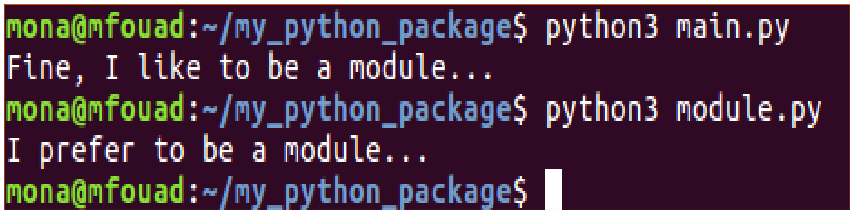
- Practice creating a module and import it...



```
mona@mfouad: ~/my_python_package
 GNU nano 2.5.3                File: module.py

if __name__ == "__main__":
        print("I prefer to be a module...")
else:
        print("Fine, I like to be a module...")


                               [ Read 5 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T
```



```
mona@mfouad: ~/my_python_package
 GNU nano 2.5.3                File: main.py

import module
```

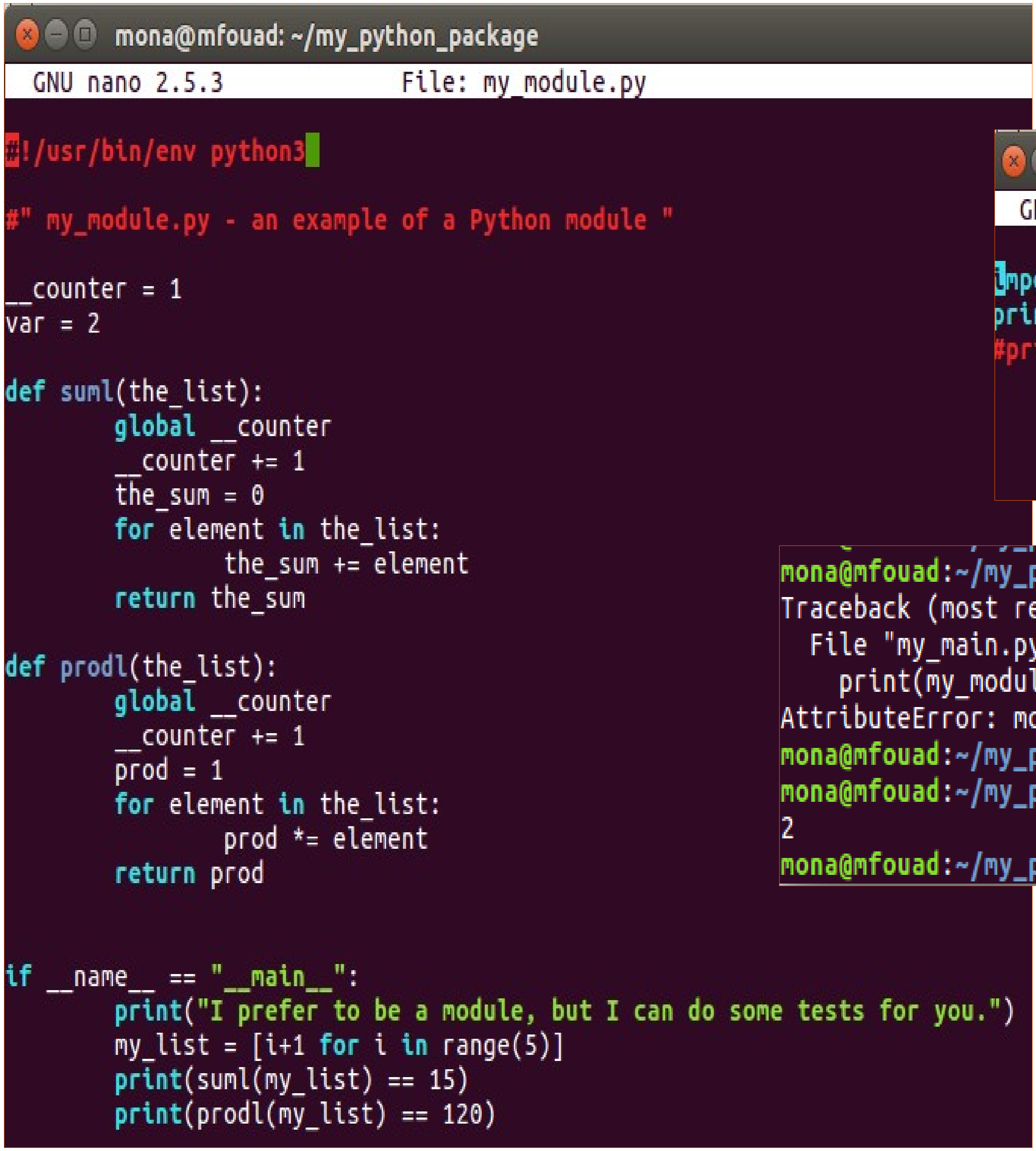


```
mona@mfouad:~/my_python_package$ python3 main.py
Fine, I like to be a module...
mona@mfouad:~/my_python_package$ python3 module.py
I prefer to be a module...
mona@mfouad:~/my_python_package$
```

Modules (cont.)

- Creating a module named my_module: Under the package directory;
  - Create a file, named my_module.py
  - Create another file, named my_functions.py that will import the module
  - Once Python imports a module, It creates a variable called __name__
  - When a file is imported as a module, its __name__ variable is set to the file's name…
  - *What will happen, if you do run the my_module.py ?*

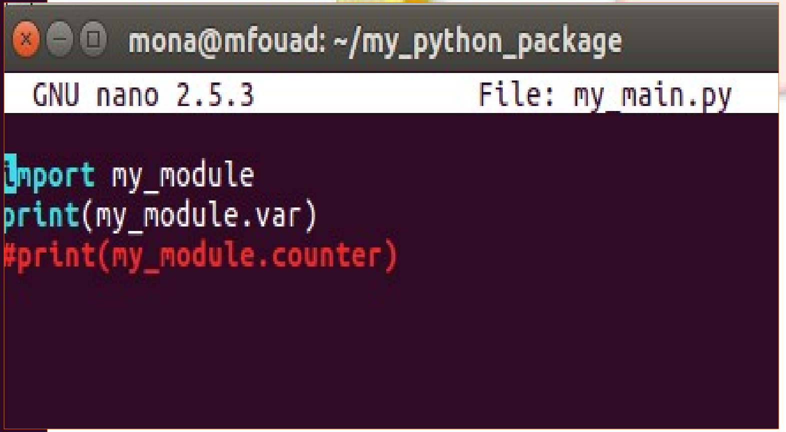
Private Variables



mona@mfouad: ~/my_python_package

GNU nano 2.5.3   File: my_module.py

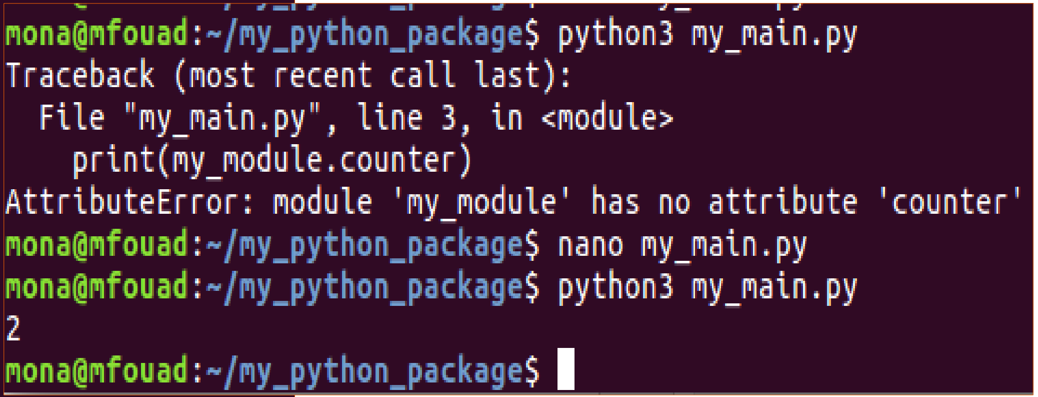
```
#!/usr/bin/env python3

#" my_module.py - an example of a Python module "

__counter = 1
var = 2

def suml(the_list):
        global __counter
        __counter += 1
        the_sum = 0
        for element in the_list:
                the_sum += element
        return the_sum

def prodl(the_list):
        global __counter
        __counter += 1
        prod = 1
        for element in the_list:
                prod *= element
        return prod


if __name__ == "__main__":
        print("I prefer to be a module, but I can do some tests for you.")
        my_list = [i+1 for i in range(5)]
        print(suml(my_list) == 15)
        print(prodl(my_list) == 120)
```



mona@mfouad: ~/my_python_package

GNU nano 2.5.3   File: my_main.py

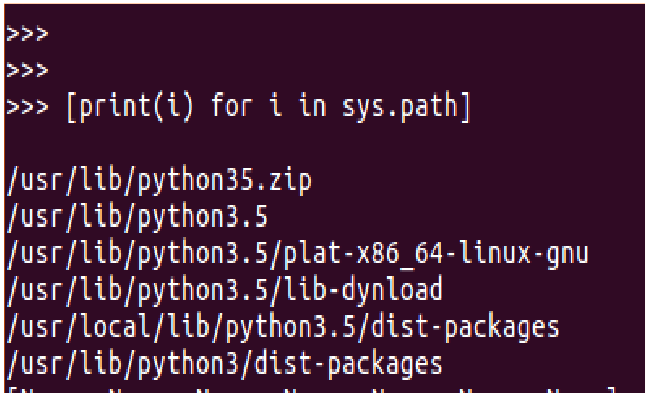
```
import my_module
print(my_module.var)
#print(my_module.counter)
```



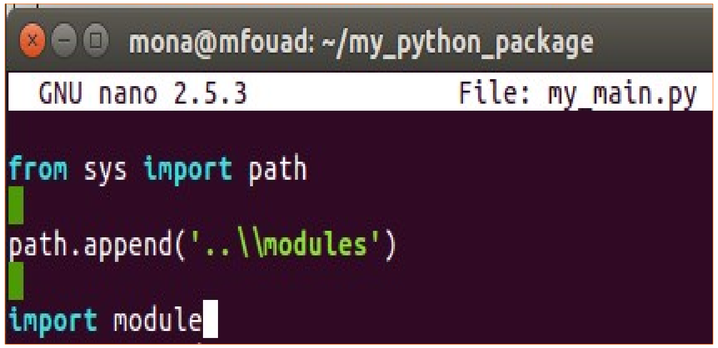
```
mona@mfouad:~/my_python_package$ python3 my_main.py
Traceback (most recent call last):
  File "my_main.py", line 3, in <module>
    print(my_module.counter)
AttributeError: module 'my_module' has no attribute 'counter'
mona@mfouad:~/my_python_package$ nano my_main.py
mona@mfouad:~/my_python_package$ python3 my_main.py
2
mona@mfouad:~/my_python_package$
```

Modules (cont.)

- The line starting with **#!** is sometimes essential for module files...
  - For Unix and Unix-like OSs (including MacOS) such a line instructs the OS how to execute the contents of the file (in other words, what program needs to be launched to interpret the text).
  - In some environments (especially those connected with web servers) the absence of that line will cause problems. This convention has no effect under MS Windows.
- Till now, both the my_module.py and the file import it (my_main.py) are located under the same folder/directory.
- If my_module.py is located under different directory than my_main.py, add the following lines before the import line



```
>>>
>>>
>>> [print(i) for i in sys.path]

/usr/lib/python35.zip
/usr/lib/python3.5
/usr/lib/python3.5/plat-x86_64-linux-gnu
/usr/lib/python3.5/lib-dynload
/usr/local/lib/python3.5/dist-packages
/usr/lib/python3/dist-packages
```



mona@mfouad: ~/my_python_package

```
 GNU nano 2.5.3                File: my_main.py

from sys import path

path.append('..\\modules')

import module
```

Solved Quiz

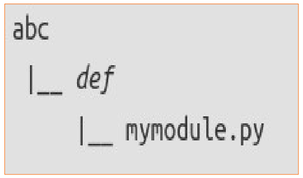
- You want to prevent your module's user from running your code as an ordinary script. How will you achieve such an effect?

```
import sys
if __name__ == "__main__":
```

- Some additional and necessary packages are stored inside the `D:\Python\Project\Modules` directory. Write a code ensuring that the directory is traversed by Python in order to find all requested modules.

```
import sys
# note the double backslashes!
sys.path.append("D:\\Python\\Project\\Modules")
  print "Don't do that!"
  sys.exit()
```

- The directory mentioned in the previous exercise contains a sub-tree of the following structure:

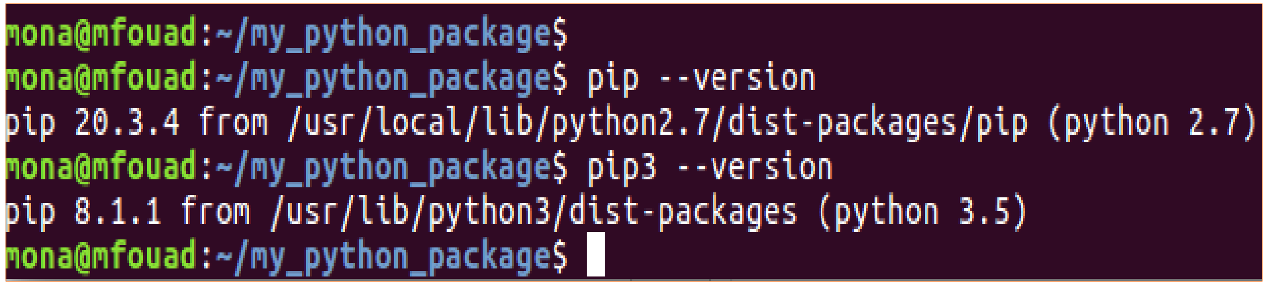


```
abc
 |__ def
      |__ mymodule.py
```

Import abc.def.my_module

Python Package Installer (PIP)

- Check the presence of pip and/or pip3

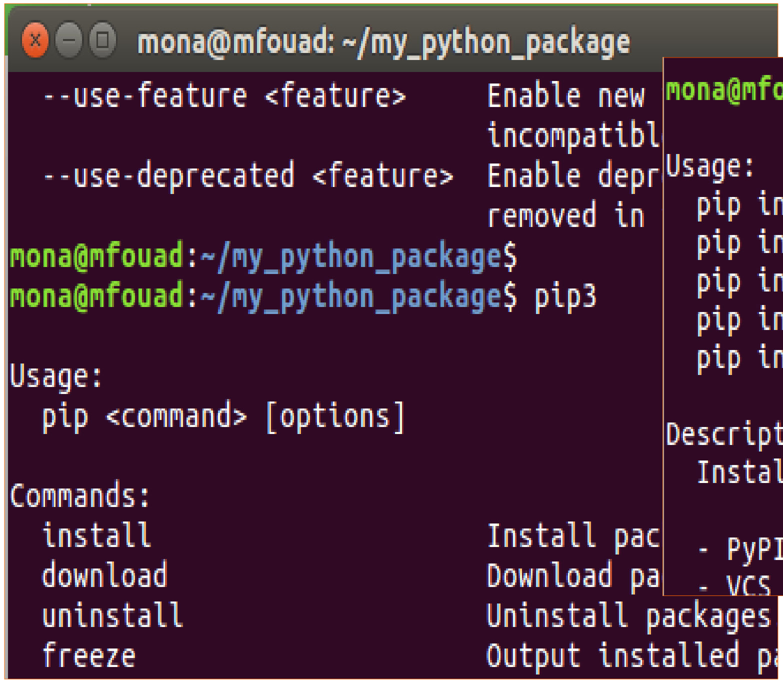


```
mona@mfouad:~/my_python_package$
mona@mfouad:~/my_python_package$ pip --version
pip 20.3.4 from /usr/local/lib/python2.7/dist-packages/pip (python 2.7)
mona@mfouad:~/my_python_package$ pip3 --version
pip 8.1.1 from /usr/lib/python3/dist-packages (python 3.5)
mona@mfouad:~/my_python_package$ 
```

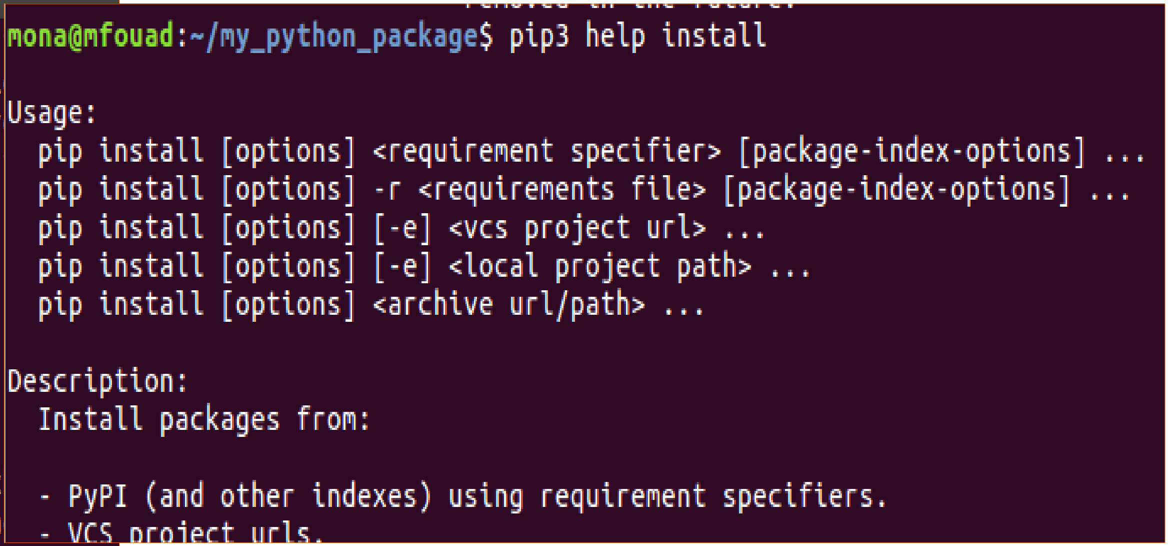
- Dependencies—certain package may depend on other(s).
- pip can discover, identify, and resolve all dependencies.
- Moreover, it can do it in the cleverest way, avoiding any unnecessary downloads and re-installs.

Python Package Installer (cont.)

- How to use pip or pip3



```
mona@mfouad: ~/my_python_package
  --use-feature <feature>     Enable new
                              incompatibl
  --use-deprecated <feature>  Enable depr
                              removed in
mona@mfouad:~/my_python_package$
mona@mfouad:~/my_python_package$ pip3

Usage:
  pip <command> [options]

Commands:
  install                     Install pac
  download                    Download pa
  uninstall                   Uninstall packages
  freeze                      Output installed pa
```



```
                              removed in the future.
mona@mfouad:~/my_python_package$ pip3 help install

Usage:
  pip install [options] <requirement specifier> [package-index-options] ...
  pip install [options] -r <requirements file> [package-index-options] ...
  pip install [options] [-e] <vcs project url> ...
  pip install [options] [-e] <local project path> ...
  pip install [options] <archive url/path> ...

Description:
  Install packages from:

  - PyPI (and other indexes) using requirement specifiers.
  - VCS project urls.
```

- If any of the currently installed packages are no longer needed ... *`pip uninstall package_name`*

Python Support

- https://docs.python.org/3/tutorial/
- https://www.w3schools.com/python/
- ...
- At Python prompt, type help(module or object)

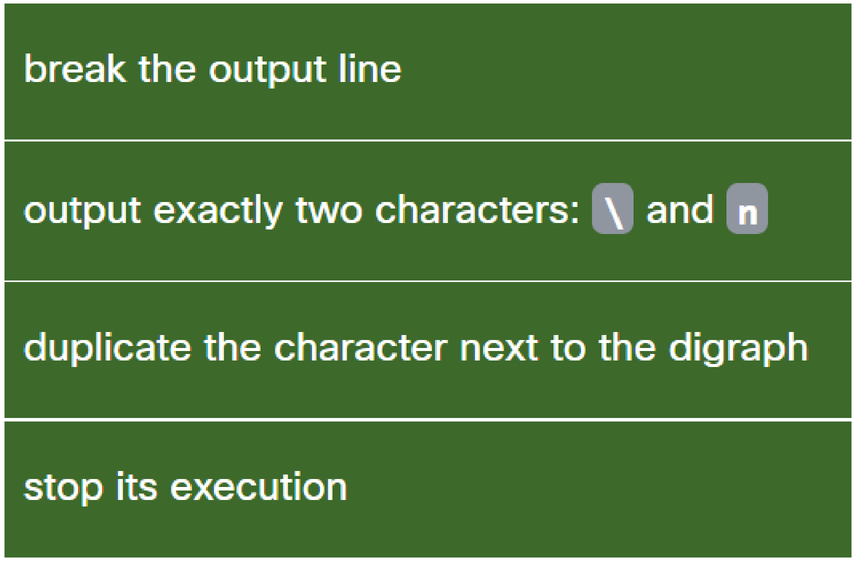
Python for Security

Knowledge Check

Quiz 1

Cisco Networking Academy

- The \n digraph forces the print() function to:



break the output line

output exactly two characters: `\` and `n`

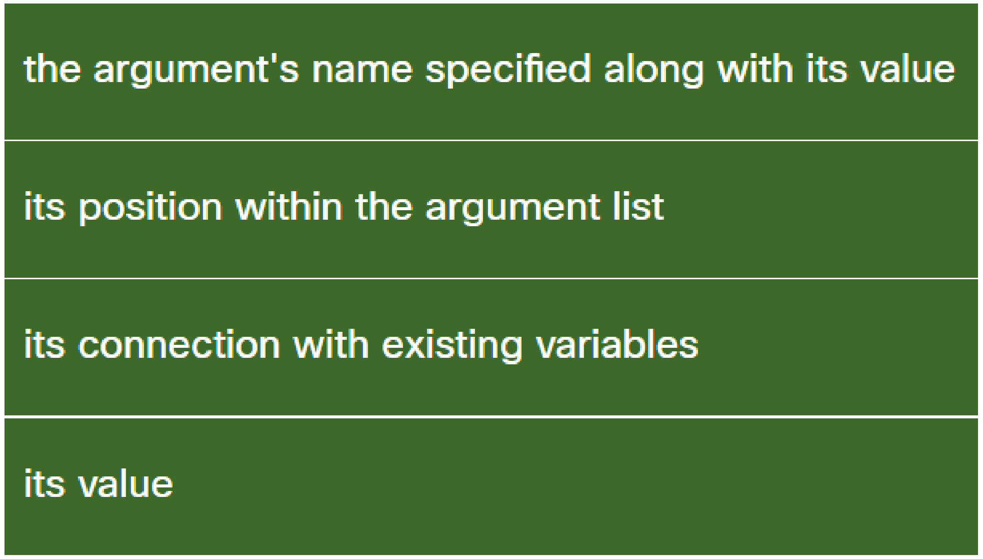
duplicate the character next to the digraph

stop its execution

Quiz 2

Cisco Networking Academy

- The meaning of the keyword parameter is determined by:



the argument's name specified along with its value

its position within the argument list

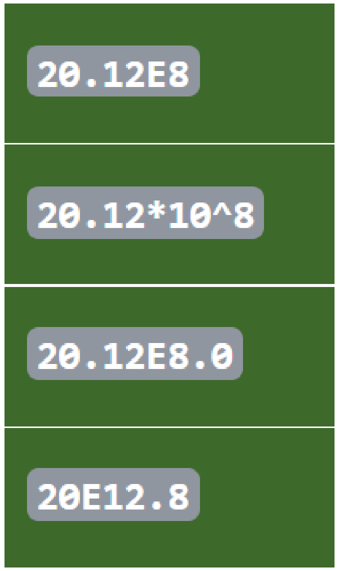
its connection with existing variables

its value

Quiz 3

Cisco Networking Academy

- The value twenty point twelve times ten raised to the power of eight should be written as:



```
20.12E8
```

```
20.12*10^8
```

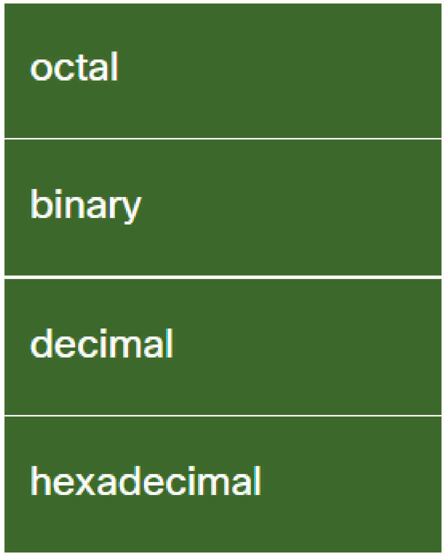
```
20.12E8.0
```

```
20E12.8
```

Quiz 4

Cisco Networking Academy

- The 0o prefix means that the number after it is denoted as:



octal

binary

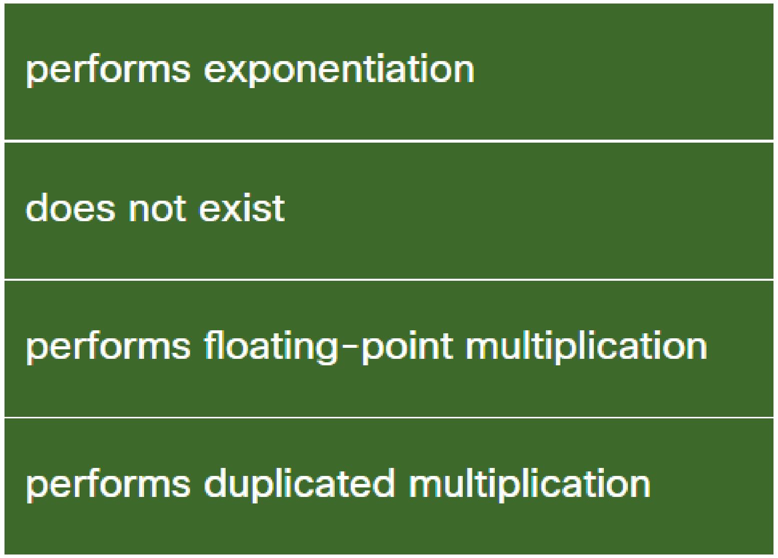
decimal

hexadecimal

Quiz 5

Cisco Networking Academy

- The ** operator:



performs exponentiation

does not exist

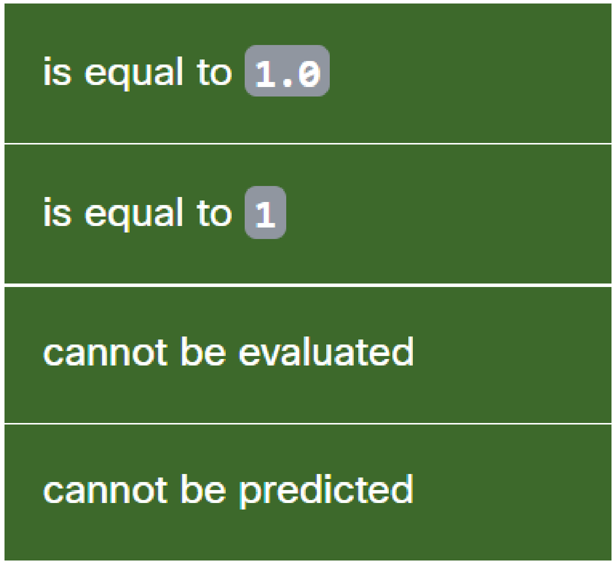
performs floating-point multiplication

performs duplicated multiplication

Quiz 6

Cisco Networking Academy

- The result of the following division: 1 / 1



is equal to `1.0`

is equal to `1`

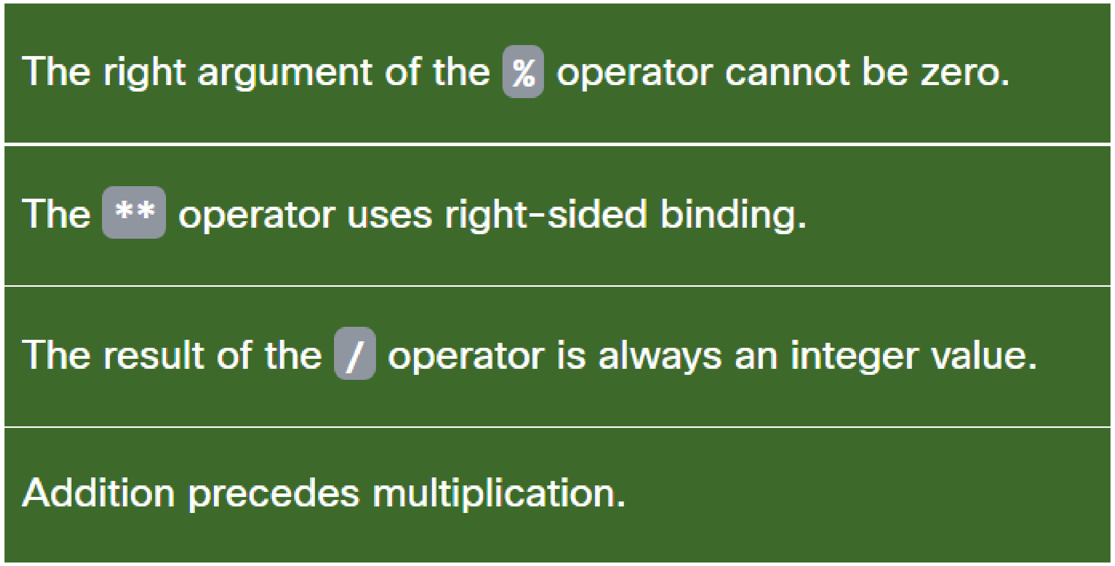
cannot be evaluated

cannot be predicted

Quiz 7

Cisco Networking Academy

- Which of the following statements are true? (Select two answers)



The right argument of the % operator cannot be zero.

The ** operator uses right-sided binding.

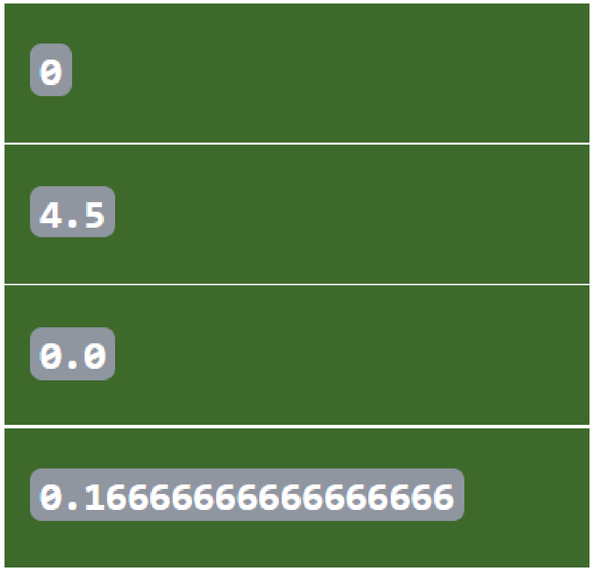
The result of the / operator is always an integer value.

Addition precedes multiplication.

Quiz 8

Cisco Networking Academy

- Left-sided binding determines that the result of the following expression: 1 // 2 * 3



```
0
```

```
4.5
```

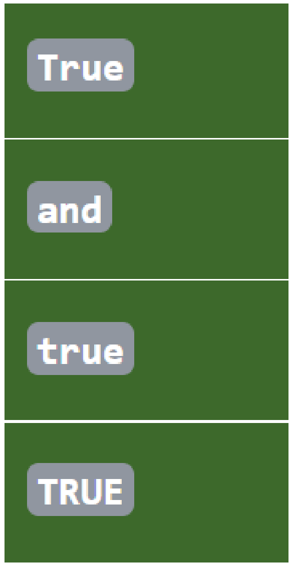
```
0.0
```

```
0.16666666666666666
```

Quiz 9

Cisco Networking Academy

- Which of the following variable names are illegal? (Select two answers)



`True`

`and`

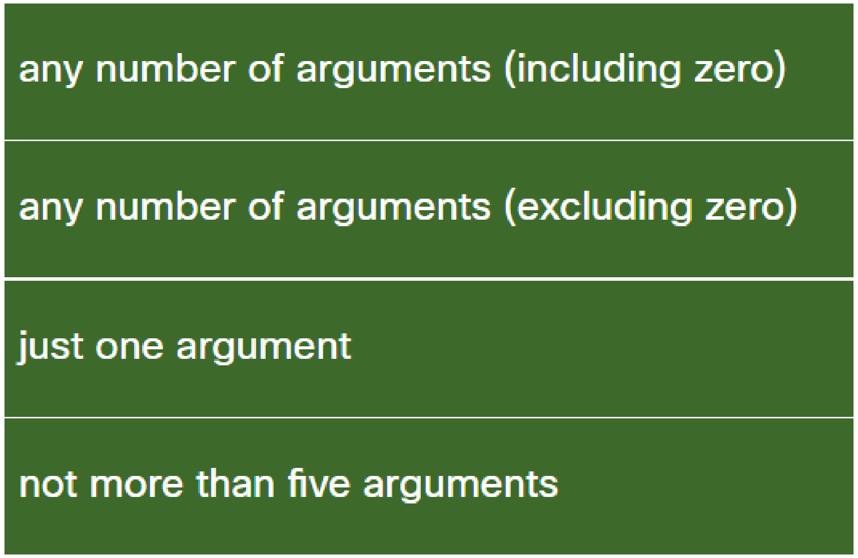
`true`

`TRUE`

Quiz 10

Cisco Networking Academy

- The print() function can output values of:

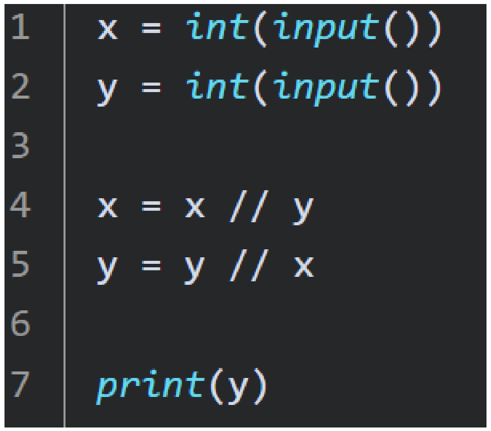


any number of arguments (including zero)
any number of arguments (excluding zero)
just one argument
not more than five arguments

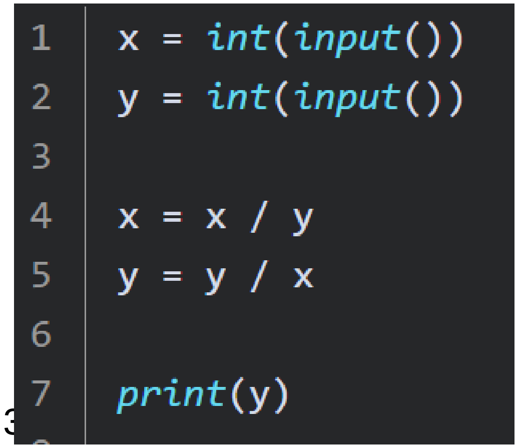
Quiz 11

Cisco Networking Academy

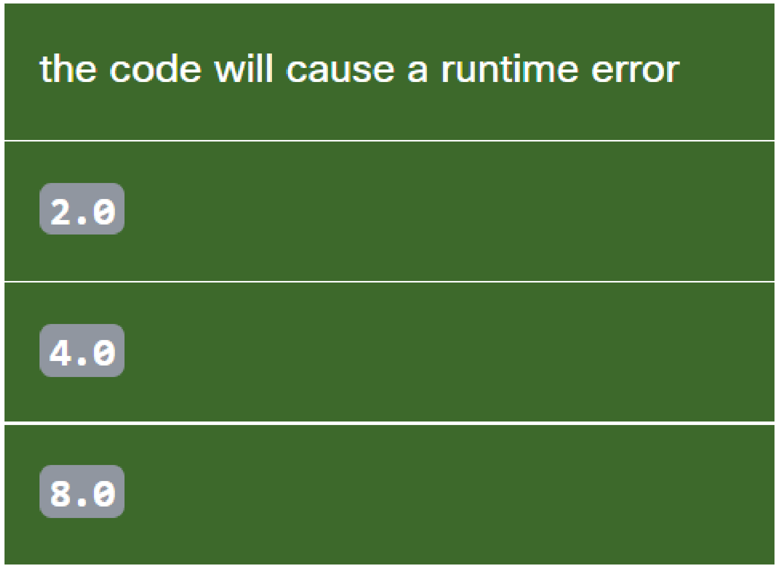
- What is the output of the following snippets if the user enters two lines containing 2 and 4 respectively?



```python
x = int(input())
y = int(input())

x = x // y
y = y // x

print(y)
```



```python
x = int(input())
y = int(input())

x = x / y
y = y / x

print(y)
```



the code will cause a runtime error

`2.0`

`4.0`

`8.0`

Quiz 12

Cisco Networking Academy

- What is the output of the following snippet?

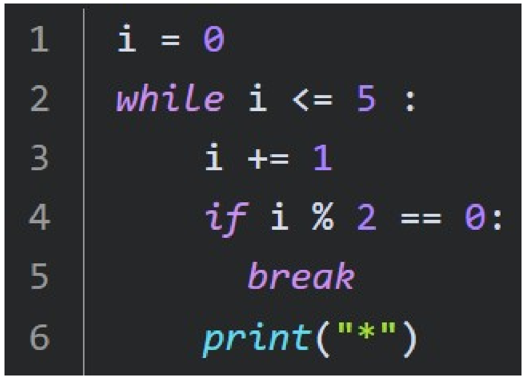
```python
x = 1 / 2 + 3 // 3 + 4 ** 2
print(x)
```

17.5
17
8
8.5

Quiz 13

Cisco Networking Academy

- How many stars (*) will the following snippet send to the console?



```python
i = 0
while i <= 5 :
    i += 1
    if i % 2 == 0:
        break
    print("*")
```

one

two

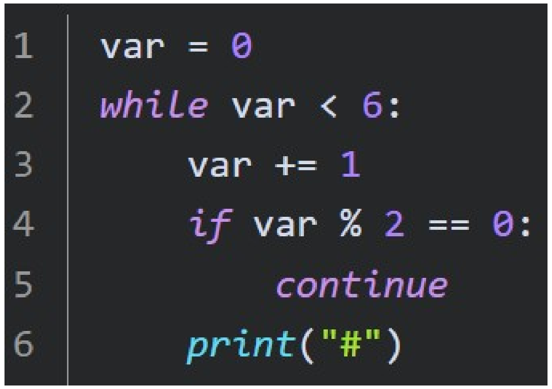
three

zero

Quiz 14

Cisco Networking Academy

- How many hashes (#) will the following snippet send to the console?



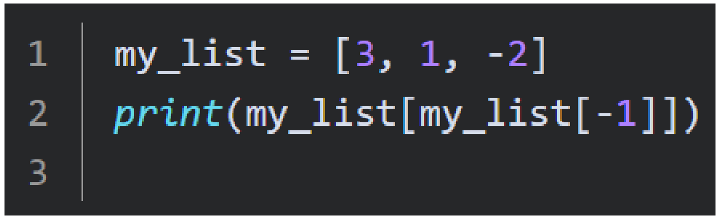
```
var = 0
while var < 6:
    var += 1
    if var % 2 == 0:
        continue
    print("#")
```

three
zero
one
two

Quiz 15

Cisco Networking Academy

- What is the output of the following snippet?



```python
my_list = [3, 1, -2]
print(my_list[my_list[-1]])
```

1

3

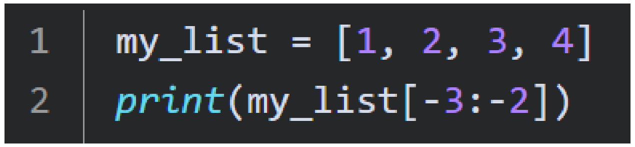
-2

-1

Quiz 15

Cisco Networking Academy

- What is the output of the following snippet?



```python
my_list = [1, 2, 3, 4]
print(my_list[-3:-2])
```

`[2]`

`[]`

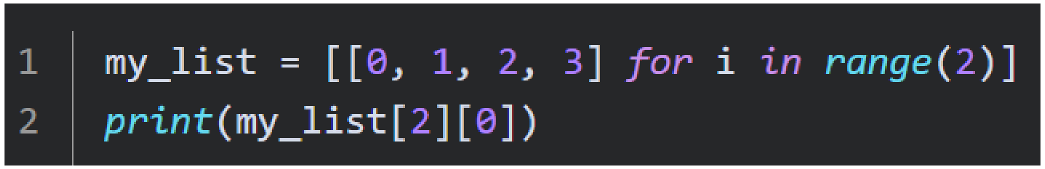
`[2, 3]`

`[2, 3, 4]`

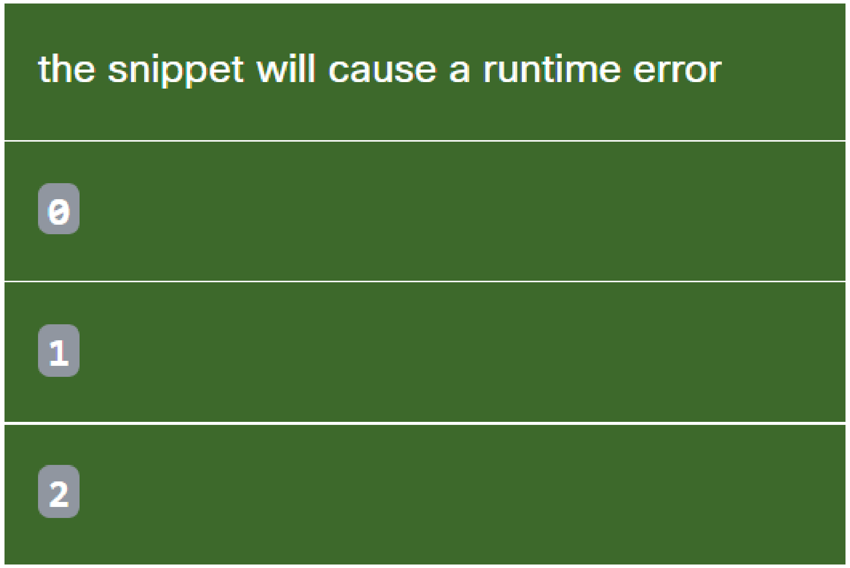
Quiz 16

Cisco Networking Academy

- What is the output of the following snippet?



```python
my_list = [[0, 1, 2, 3] for i in range(2)]
print(my_list[2][0])
```



the snippet will cause a runtime error

0

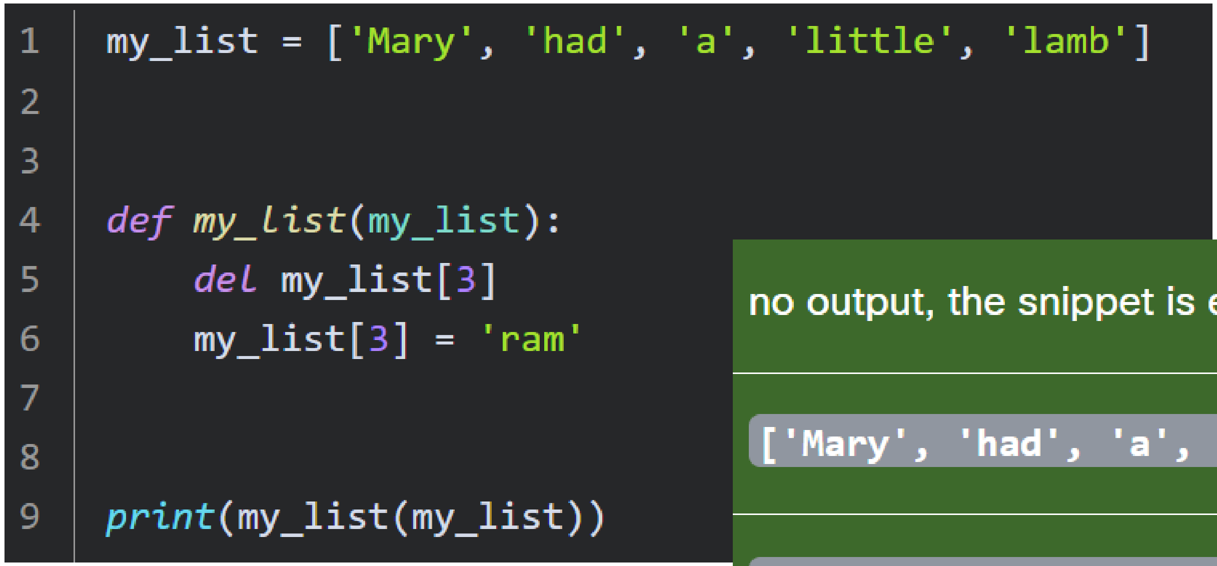
1

2

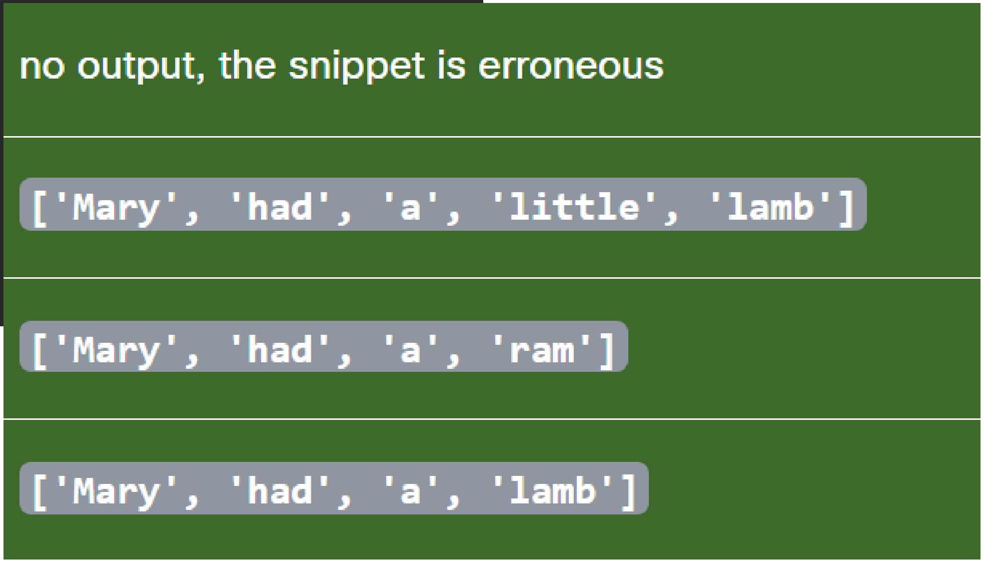
Quiz 17

Cisco Networking Academy

- What is the output of the following snippet?



```python
my_list = ['Mary', 'had', 'a', 'little', 'lamb']


def my_list(my_list):
    del my_list[3]
    my_list[3] = 'ram'


print(my_list(my_list))
```



no output, the snippet is erroneous

`['Mary', 'had', 'a', 'little', 'lamb']`

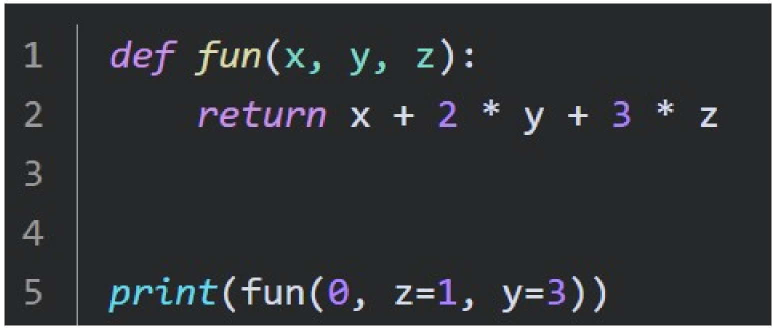
`['Mary', 'had', 'a', 'ram']`

`['Mary', 'had', 'a', 'lamb']`

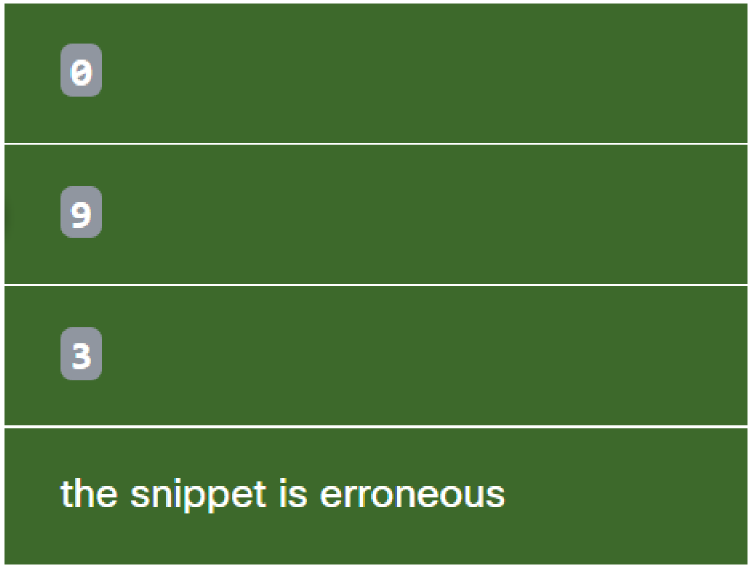
Quiz 18

Cisco Networking Academy

- What is the output of the following snippet?



```python
def fun(x, y, z):
    return x + 2 * y + 3 * z


print(fun(0, z=1, y=3))
```



0

9

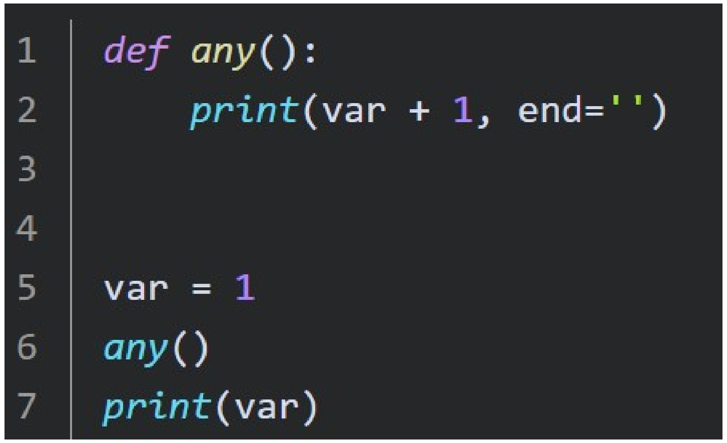
3

the snippet is erroneous

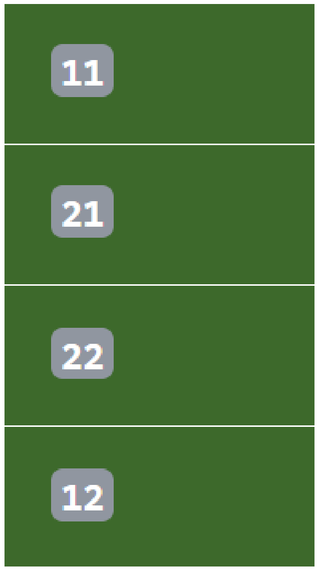
Quiz 19

Cisco Networking Academy

- What is the output of the following snippet?



```
def any():
    print(var + 1, end='')


var = 1
any()
print(var)
```



11

21

22

12

Quiz 20

Cisco Networking Academy

- Which of the following lines properly starts a function using two parameters, both with zeroed default values?

```
def fun(a=0, b=0):
```

```
fun fun(a=0, b):
```

```
def fun(a=b=0):
```

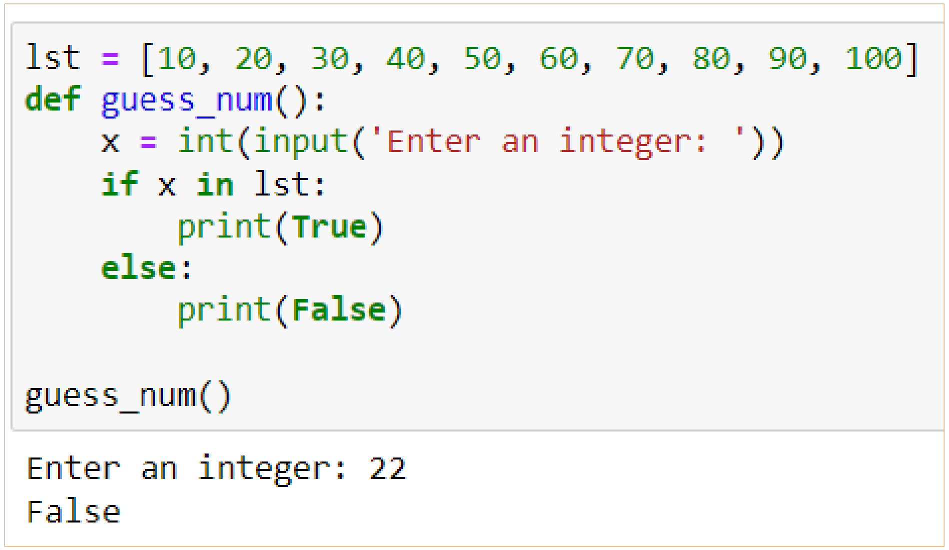
```
fun fun(a, b=0):
```

Programming Task

1) ***Create a Python code*** that has a list of random digits and receive a number from user and check whether s(he) does correctly guess a number from your list or not...develop the needed functions for that purpose.

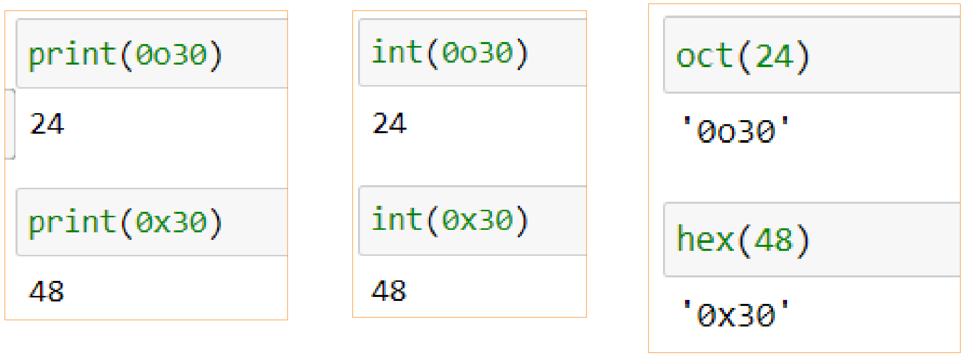
2) ***Self study activity:*** print out hexadecimal and octet numbers...try the numbers 2, 7, 20, 30, ...

Programming Task (solution)

- 

```
lst = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
def guess_num():
    x = int(input('Enter an integer: '))
    if x in lst:
        print(True)
    else:
        print(False)

guess_num()
```

```
Enter an integer: 22
False
```

- 

```
print(0o30)
```

```
24
```

```
print(0x30)
```

```
48
```

```
int(0o30)
```

```
24
```

```
int(0x30)
```

```
48
```

```
oct(24)
```

```
'0o30'
```

```
hex(48)
```

```
'0x30'
```