Module 2

- Data Types
- Variables
- Basic Input and Output
- Basic Operators
- Type conversion

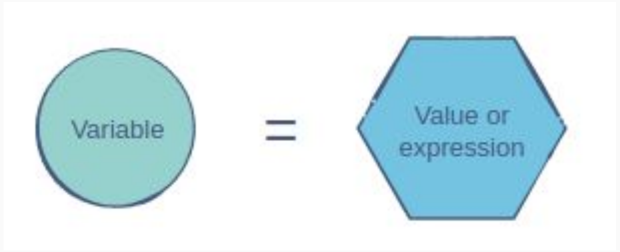
Values & Data Types

Values and Data Types

- A **value** is one of the fundamental things — like a word or a number — that a program manipulates
- A value can be any of the following - 5, “Hello World”, “23”, 15.7, etc.
- These values are often referred to as objects so values and objects can be used interchangeably
- These objects are classified into different classes or data types

Variables

- A variable is simply a name to which a value can be assigned
- It allows us to give meaningful names to data
- We use the assignment operator “ = ” to assign a value to a variable
- Variables allow us to store data for later use
- Variables are **mutable** (meaning the value of a variable can always be updated or replaced)



Variables - Naming Convention

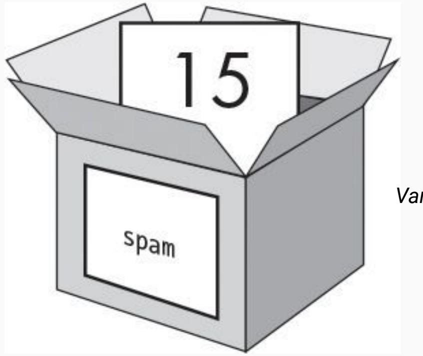
Certain rules must be followed when picking the name of a variable

- Name can start with upper or lower case alphabet - e.g country or Country
- Names are case sensitive - Country and country are two different variables not one.
- A number can appear in the name, but not at the beginning - country23, cou23try are both valid. 23country is not
- The "_" character can appear anywhere in the name e.g _country or country_ are valid

Variables - Naming Convention

Certain rules must be followed when picking the name of a variable

- Spaces are not allowed in variable names.
- snake_case can be used for variable names with multiple words. e.g asian_country
- The variable name should be meaningful - describing the value it holds.



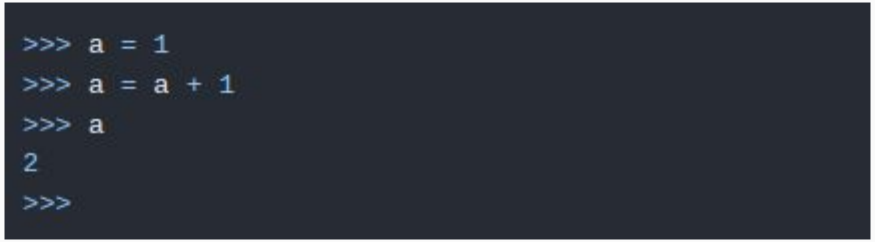
Variables are like boxes that can hold values

Variables - Keep in mind

- Variables always point to a value, **they never point to other variables**
- Multiple variables can point to the same value, but one variable cannot point to multiple values
- The variable name should be meaningful - describing the value it holds.
- The values that variables point to can point to other values also
- Some words called **keywords** words can not be used as variables. They are reserved to Python itself and have special meaning. Run help (‘keywords’) in the terminal to see these words

Variables - The “ = “

- When assigning something to a variable using a =, the right side of the = is always executed before the left side.
- This means that we can do something with a variable on the right side, then assign the result back to the same variable on the left side.



```
>>> a = 1
>>> a = a + 1
>>> a
2
>>>
```

https://www.youtube.com/watch?v=G41G_PEWFjE

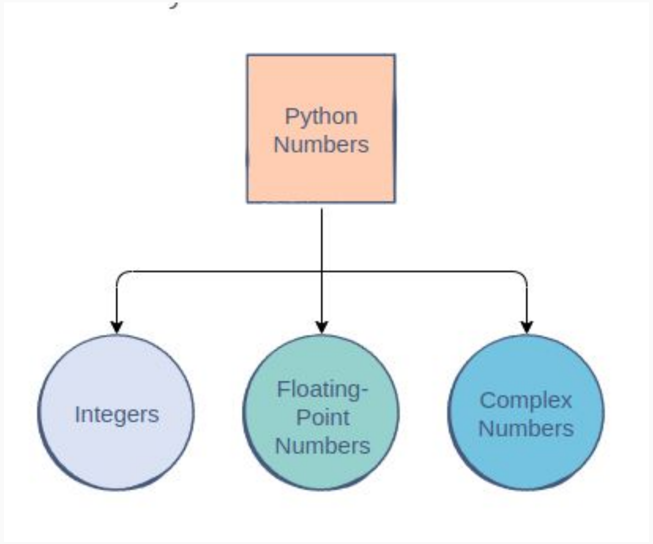
Variables

Data Types

- The data type of an item defines the type and range of values that item can have.
- There are three main data types in Python:
    - 1. Numbers
    - 2. Strings
    - 3. Booleans

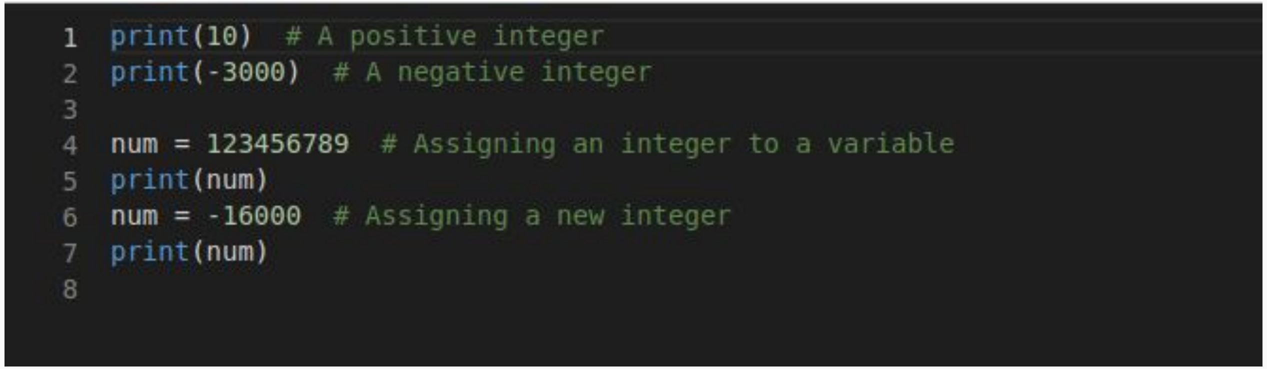
Numbers

- Python is very powerful in manipulating numerical data
- There are 3 main types of numbers in python
  - Integers
  - Floating Point Numbers
  - Complex Numbers

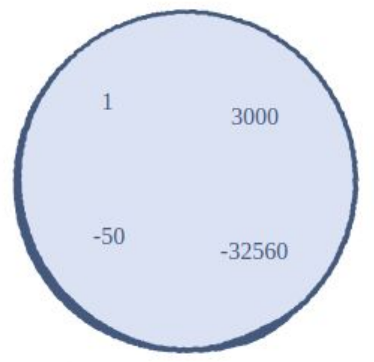


Numbers - Integers

- Integer data type is comprised of all the positive and negative whole numbers
- Integers do not have a fractional part
- All negative numbers start with the symbol "-"



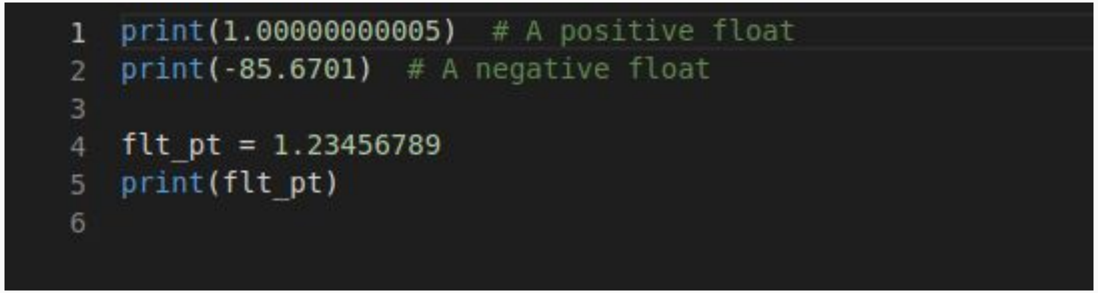
```
print(10)  # A positive integer
print(-3000)  # A negative integer

num = 123456789  # Assigning an integer to a variable
print(num)
num = -16000  # Assigning a new integer
print(num)
```



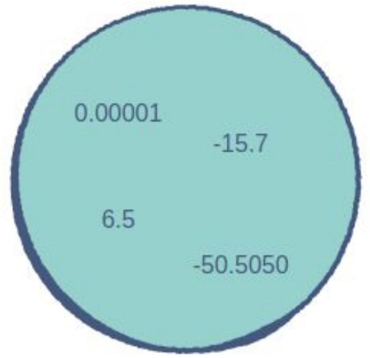
Integers in Python.

Numbers - Float

- These are numbers that have or may have a fractional part after the decimal point
- They can be positive or negative
- We can create decimals up to a very high decimal



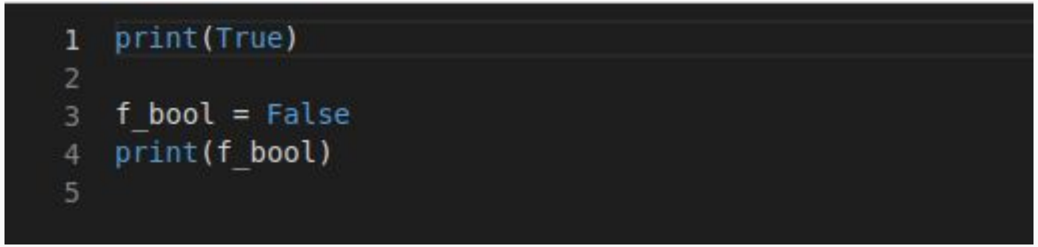
```
print(1.00000000005)  # A positive float
print(-85.6701)  # A negative float

flt_pt = 1.23456789
print(flt_pt)

```



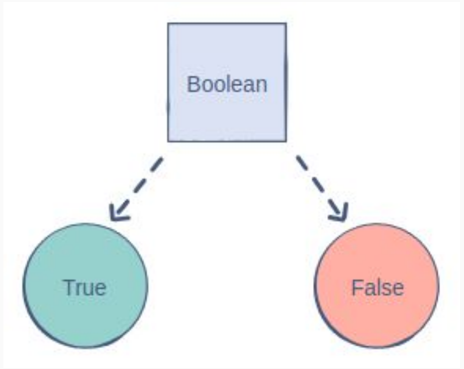
Floating-point numbers in Python.

Booleans (Bool)

- This data type allows us to choose between 2 values: **true** and **false**
- The first letter of a bool needs to capitalized: **True** or **False**
- A bool is used to determine whether the logic of an expression or a comparison is correct. (data comparisons)



```
print(True)

f_bool = False
print(f_bool)

```

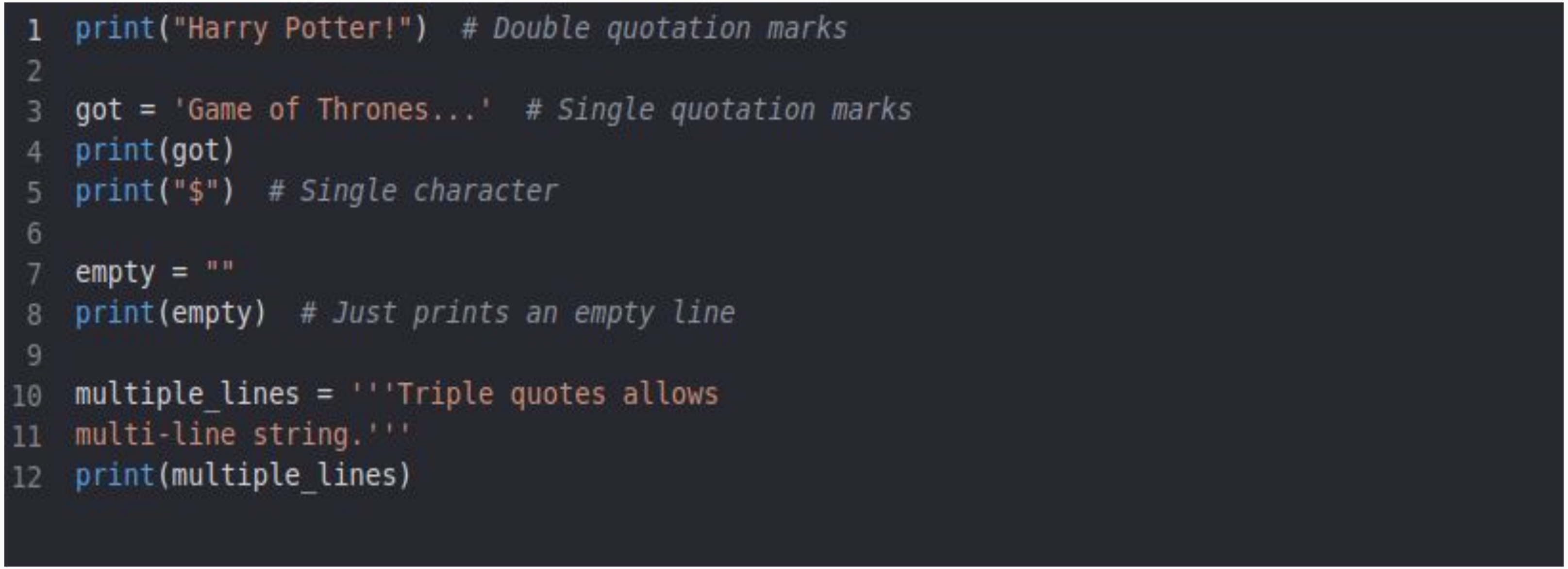


Numbers and Bool

Strings

- A collection of characters closed within single, double or triple quotation marks (‘a’, “a”, “””a“””, ‘1’, ‘&’, “gfh*56&”)
- It can contain a single character or be entirely empty
- The quotes tell python where the strings begins and ends
- Use triple quotes to enter a multiline string (‘’’ ‘’’)
- Double quoted strings can contain single quotes in them (`"Bruce's beard"`)
- Single quoted strings can have double quotes in them (`'The knights who say "Ni!"'`)

Strings



```python
print("Harry Potter!")  # Double quotation marks

got = 'Game of Thrones...'  # Single quotation marks
print(got)
print("$")  # Single character

empty = ""
print(empty)  # Just prints an empty line

multiple_lines = '''Triple quotes allows
multi-line string.'''
print(multiple_lines)
```

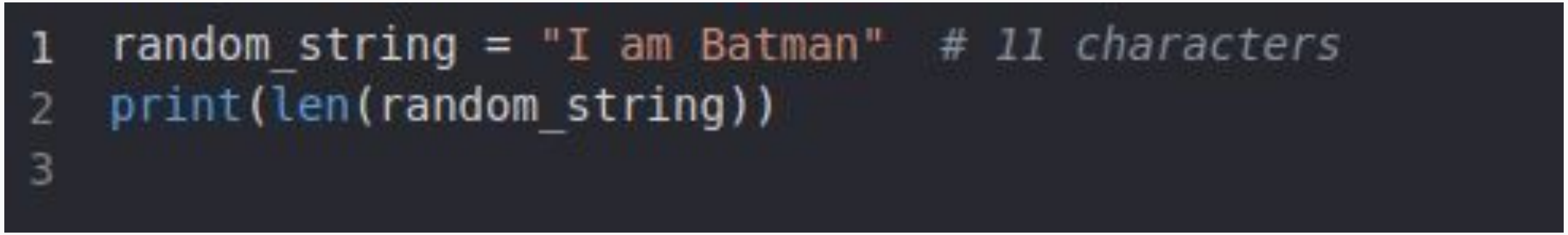
Strings - Examples

Strings can have any keyboard character in them and can be as long as you want

- 'hello'
- 'Hi there!'
- 'KITTENS'
- '7 apples, 14 oranges, 3 lemons'
- 'Anything not pertaining to elephants is irrelephant.'
- 'A long time ago, in a galaxy far, far away...'
- 'O*&#wY%*&OCfsdYO*&gfC%YO*&%3yc8r2'

The Length of a String

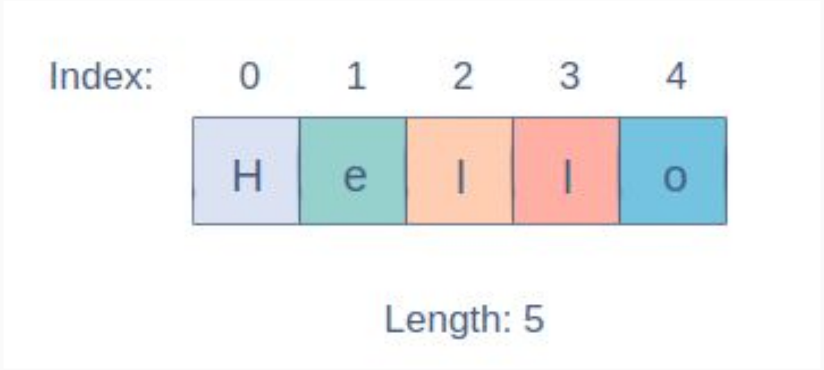
- The built-in-function len() can be used to check the length of a string
- The length of a string includes all the characters in the string



```python
random_string = "I am Batman"  # 11 characters
print(len(random_string))
```

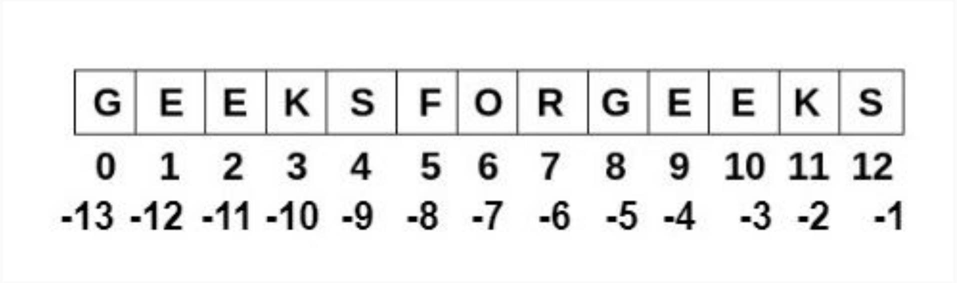
String Indexing

- Every character in a string has a numerical index based on its position
- Indexing start from 0 to n -1, where n is the length of the string
- The index of the first character of a string is 0



String - Accessing Characters (Indexing)

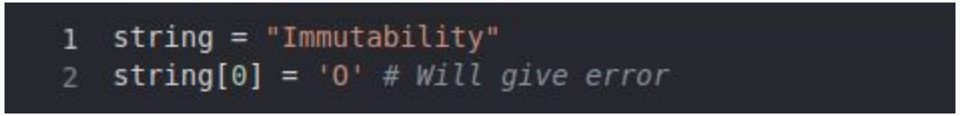
- A string character can be accessed using its index
- Index must be closed with [ ]
- Indexing can start from the beginning of a string **(positive indexing)** or at the end of the string **(negative indexing)**
- Accessing an index out of the range of the length of the string will cause an **IndexError**



G	E	E	K	S	F	O	R	G	E	E	K	S
0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

String - Immutability

- Strings are **immutable** - meaning that can not be changed in place
- Once a value is assigned to a string it can not be updated later
- Assigning a new value to a string variable creates a new string in memory
- Trying to change the character in a string to a different value will throw a **TypeError**



```python
string = "Immutability"
string[0] = 'O' # Will give error
```

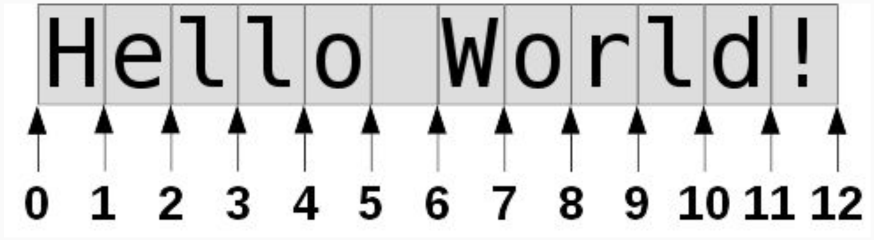
strings

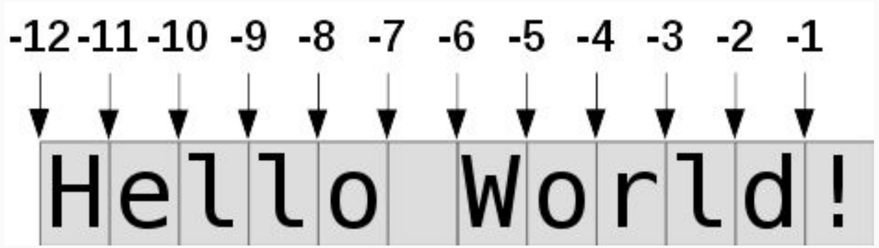
String - Slicing

- Slicing is the process of obtaining a portion (substring) of a string by using its indices
- The following template can be used to slice a string:

string [start : end]

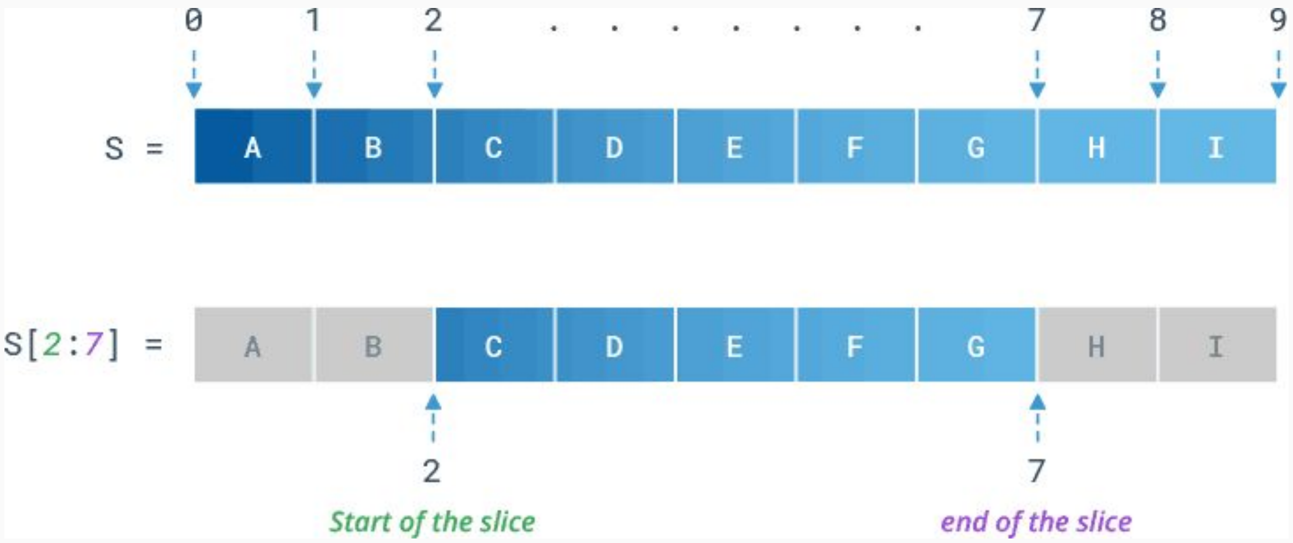
- Start is the index from where we want the substring to start
- End is the index where we want our substring to end





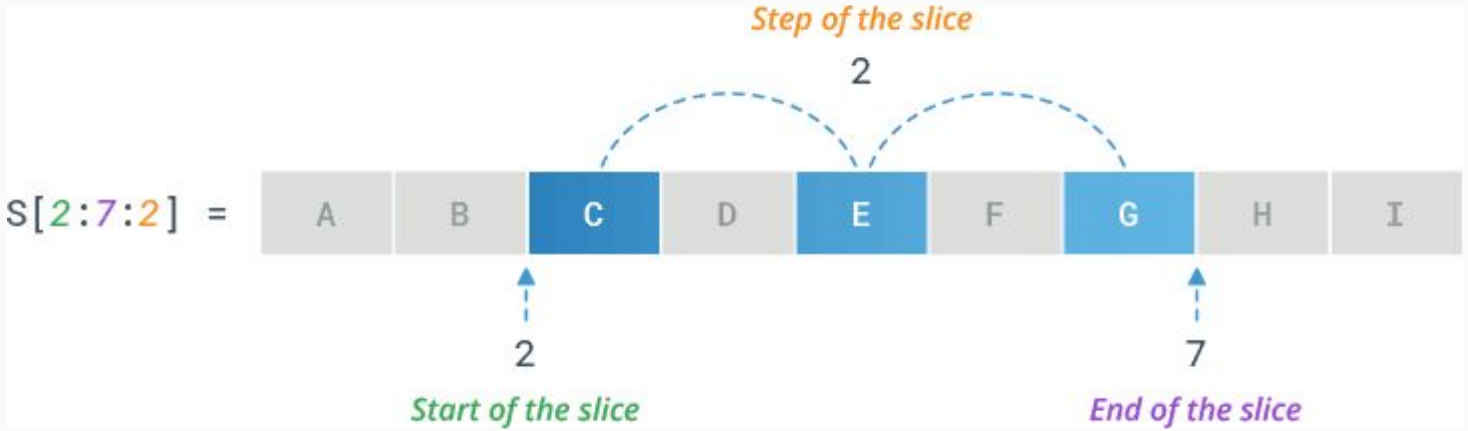
String - Slicing

- string [start : end]
- Using this method, the character at the end index in the string will not be included in the substring



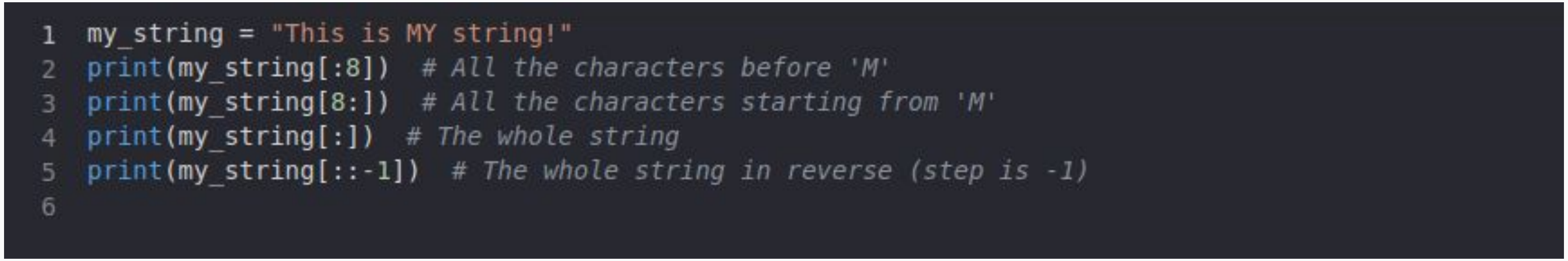
String - Slicing with a step

- string [start : end : step]
- The step is used to skip characters in the string
- The default step is 1, meaning iterate through the string one char at a time



String - Partial Slicing

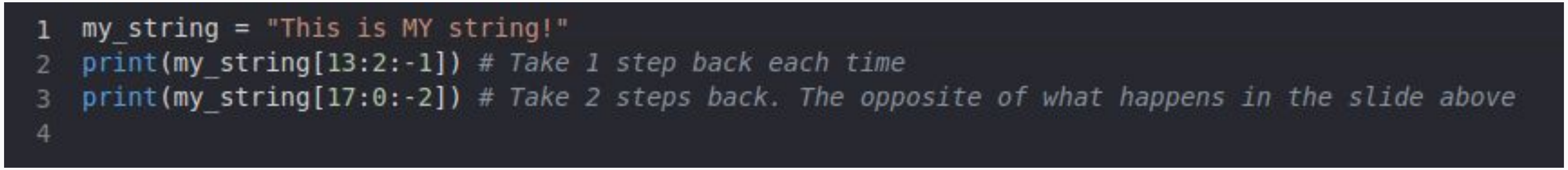
- It is actually optional to specify the start and end indices
- If the start is not provided, the substring will have all the characters until the end index
- If the end is not provided, the substring will begin from the start index and go all the away to the end



```python
my_string = "This is MY string!"
print(my_string[:8])  # All the characters before 'M'
print(my_string[8:])  # All the characters starting from 'M'
print(my_string[:])  # The whole string
print(my_string[::-1])  # The whole string in reverse (step is -1)

```

String - Reverse Slicing

- strings can also be sliced to return a reversed substring
- The order of the start and end indices get switched



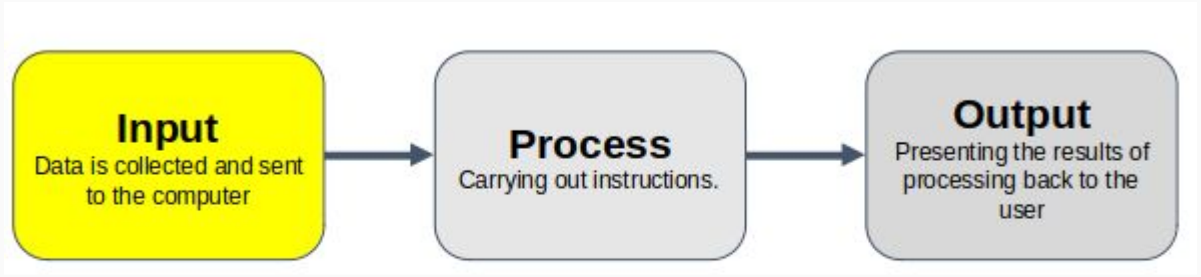
```python
my_string = "This is MY string!"
print(my_string[13:2:-1]) # Take 1 step back each time
print(my_string[17:0:-2]) # Take 2 steps back. The opposite of what happens in the slide above

```

Strings

Basic Input and Output

INPUT

- Input can be used to interact with the user
- It is used to get data from the user
- Input data **must** always be assigned to a variable


Input
Data is collected and sent to the computer
Process
Carrying out instructions.
Output
Presenting the results of processing back to the user

INPUT - Example

```
print (“Hi! What’s your name?”)

name = “Dave”

print (“Hi “ + name + “! How are you today?”)
```

```
print (“Hi! What’s your name?”)

name = input ( )

print (“Hi “ + name + “! How are you today?”)
```

INPUT - Example

```
print (“Hi! What’s your name?”)

name = “Dave”

print (“Hi “ + name + “! How are you today?”)
```

```
name = input (“What’s your name?” ) # more efficient

print (“Hi “ + name + “! How are you today?”)
```

Input

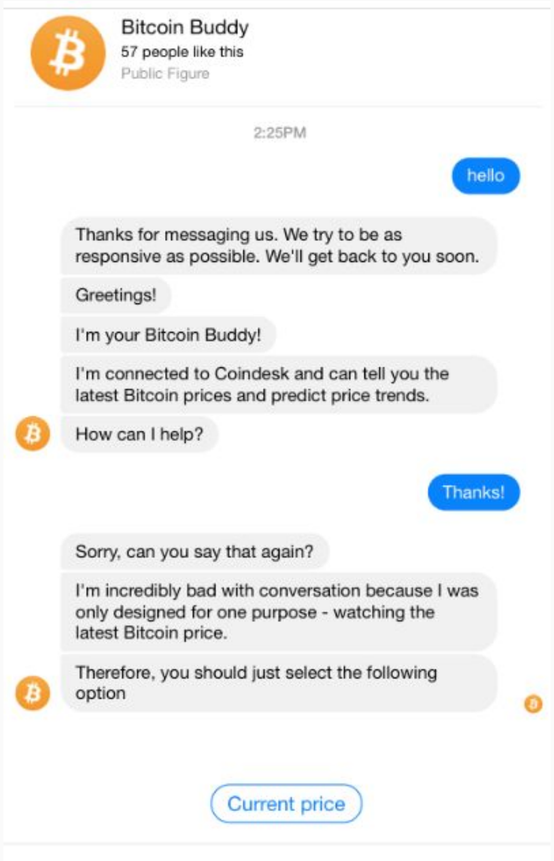
Assignment -VARIABLES | INPUT | STRINGS

Chat-Bot Challenge

Lot of websites use chat bots to interact with their customers . These chat bots are often very sophisticated and use AI to learn and adapt to the user. Our chat bot is going to be simpler.

The chat bot should work like this:

- Ask the user their name and store it in a variable.
- Greet the user by name.
- Ask the user three questions about themselves and store their responses in three different suitably named variables
- Respond to each of the questions one by one, using the user's name in the response.
- Output a summary of all the user's answer in a single sentence.



Bitcoin Buddy
57 people like this
Public Figure

2:25PM

hello

Thanks for messaging us. We try to be as responsive as possible. We'll get back to you soon.

Greetings!

I'm your Bitcoin Buddy!

I'm connected to Coindesk and can tell you the latest Bitcoin prices and predict price trends.

How can I help?

Thanks!

Sorry, can you say that again?

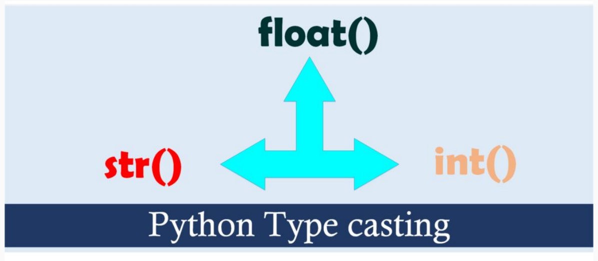
I'm incredibly bad with conversation because I was only designed for one purpose - watching the latest Bitcoin price.

Therefore, you should just select the following option

Type Conversion

Type Conversion

- Sometimes we need to convert values from one data type to another.
- Some built-in functions also known as **type conversion functions** allow us to that - int(), float() and str()
- These functions attempt to convert their arguments into *int*, *float* and *str* respectively.

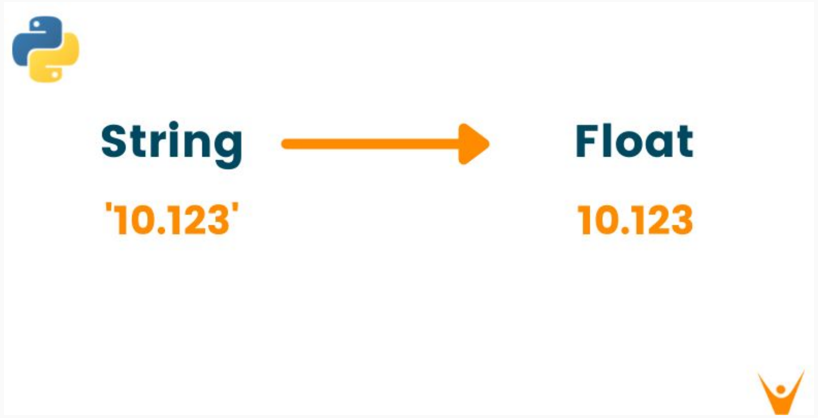


Type Conversion - int ()

- The int() function takes a *floating point number* or a *string* and turn it into an *int*.
- For floating point numbers, it discards the decimal portion of the number
- The string has to be a syntactically legal number
- That process is called ***truncation towards zero***

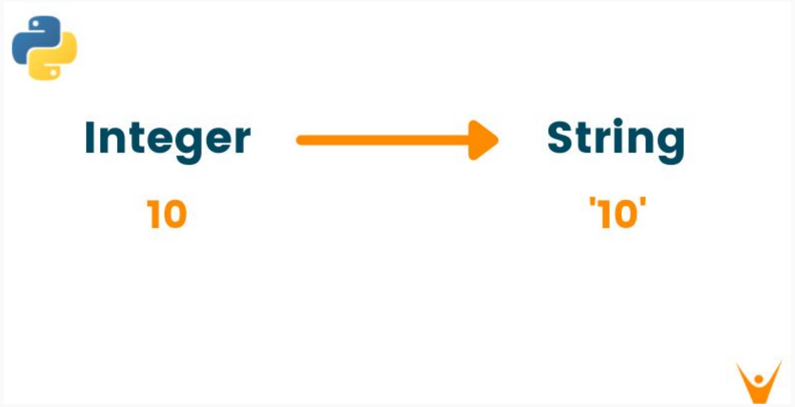
Type Conversion - float ()

- The float() function can turn an integer, a float or a syntactically legal string into a float



Type Conversion - str()

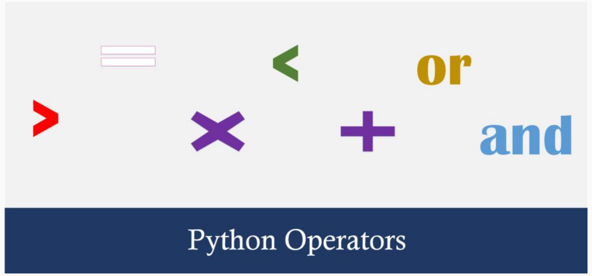
- The str() function can turn its arguments into a string



Basic Operators

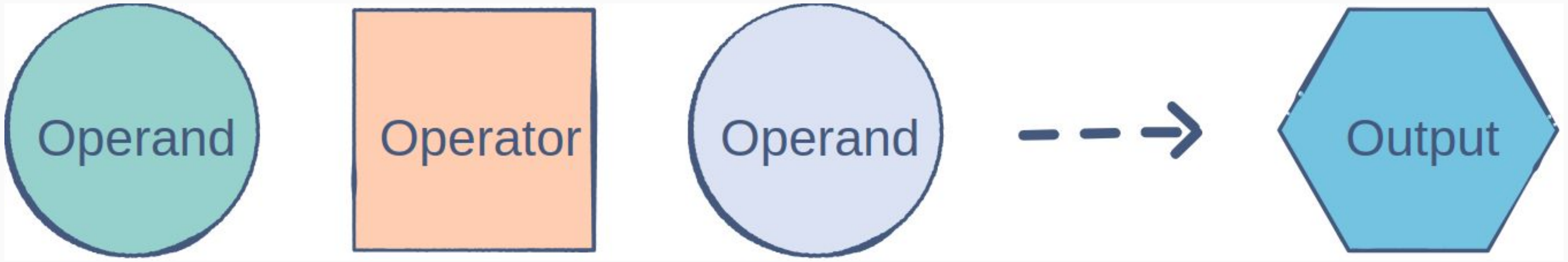
Basic Operators

- An operator is a character or special keyword, which is able to operate on the values
- Operators are used to perform arithmetic and logical operations on data
- We use them to manipulate and interpret data to produce useful output
- Python operators follow the **in-fix** or **prefix** notations



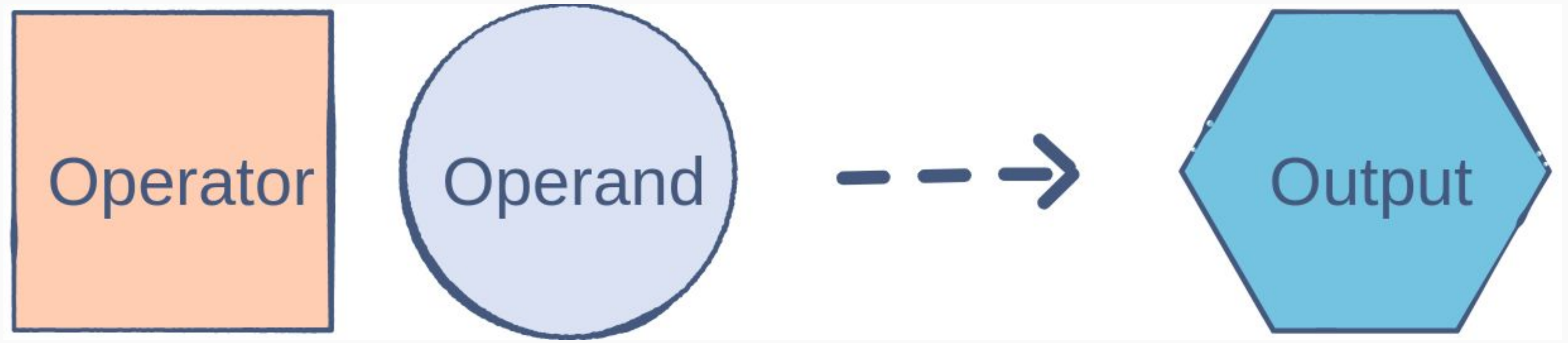
Basic Operators in-fix

- The values operators work on are called **operands**
- **In-fix** operators appear between two operands and hence are known as binary operators



Basic Operators prefix

- **prefix** operators usually work on one operand and appears before it
- They are known as unary operators



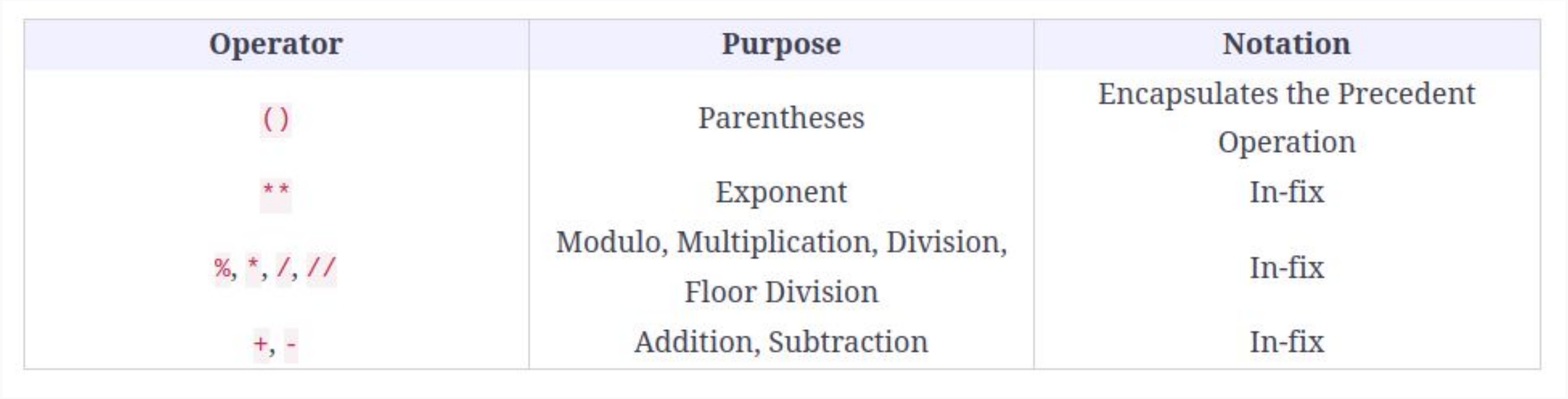
Unary and Binary Operators in python

Main Operators in Python

- arithmetic operators
- comparison operators
- assignment operators
- logical operators
- bitwise operators

Arithmetic Operators

- Here are the basic arithmetic operators in order of **precedence**



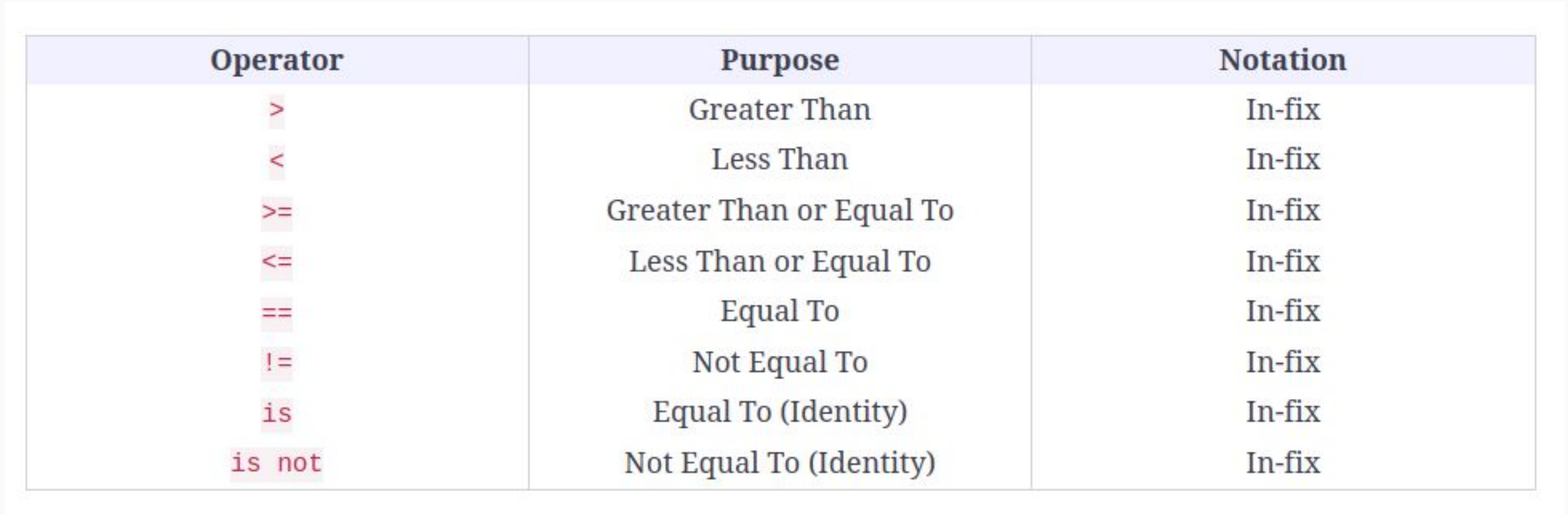
Operator	Purpose	Notation
`()`	Parentheses	Encapsulates the Precedent Operation
`**`	Exponent	In-fix
`%`, `*`, `/`, `//`	Modulo, Multiplication, Division, Floor Division	In-fix
`+`, `-`	Addition, Subtraction	In-fix

Demo Arithmetic Operators

- Addition
- Subtraction
- Multiplication
- Division
- Floor Division
- Modulo
- Precedence
- Parentheses

Comparison Operators

- These operators can be used to compare values in mathematical forms
- The result of a comparison is always a bool.



Operator	Purpose	Notation
`>`	Greater Than	In-fix
`<`	Less Than	In-fix
`>=`	Greater Than or Equal To	In-fix
`<=`	Less Than or Equal To	In-fix
`==`	Equal To	In-fix
`!=`	Not Equal To	In-fix
`is`	Equal To (Identity)	In-fix
`is not`	Not Equal To (Identity)	In-fix

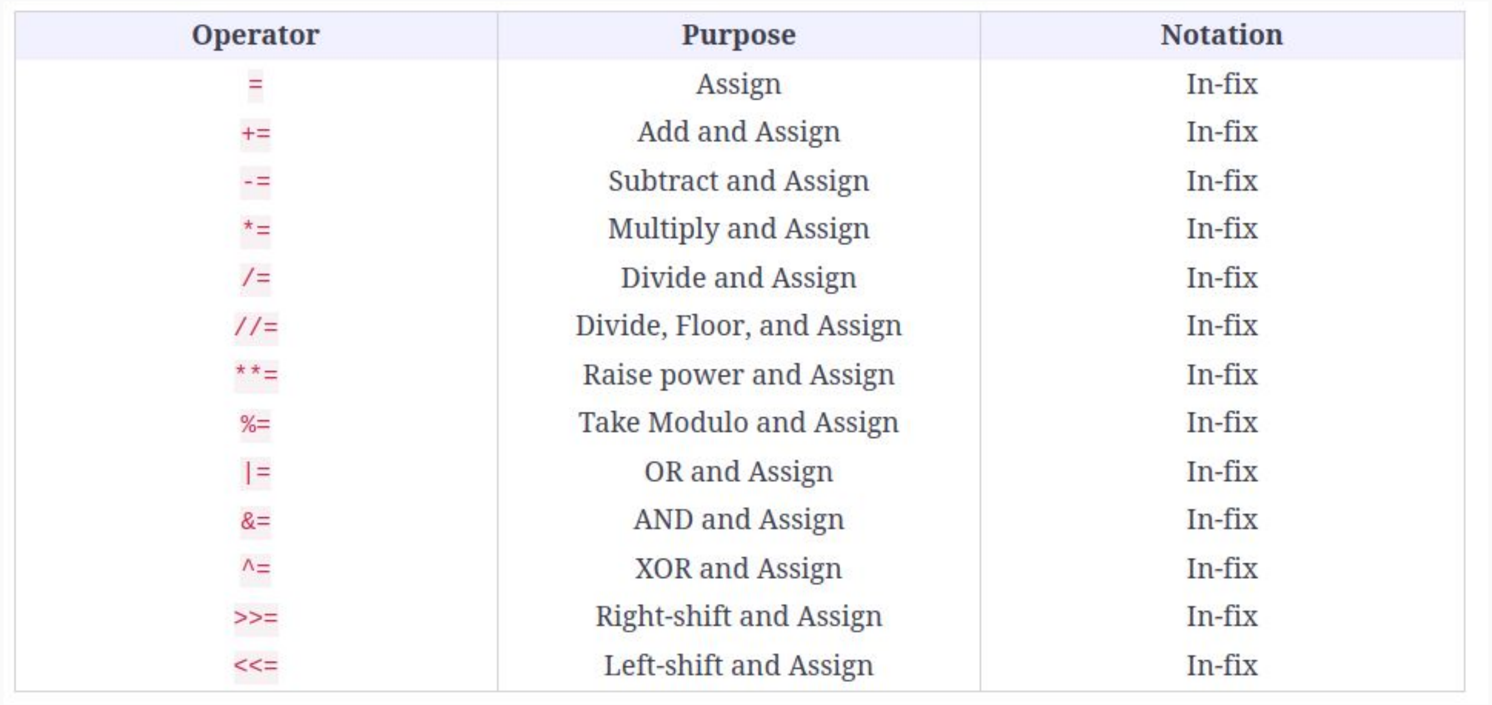
Comparison Operators

- The == and != operators compare the **values of both operands.**
- However, the identity operators, is and is not, check whether the **two operands are the exact same object in memory.**

Demo Comparison Operators

Assignment Operators

- Used to assign values to a variable.
- The = is an assignment operator, but there are others as well:

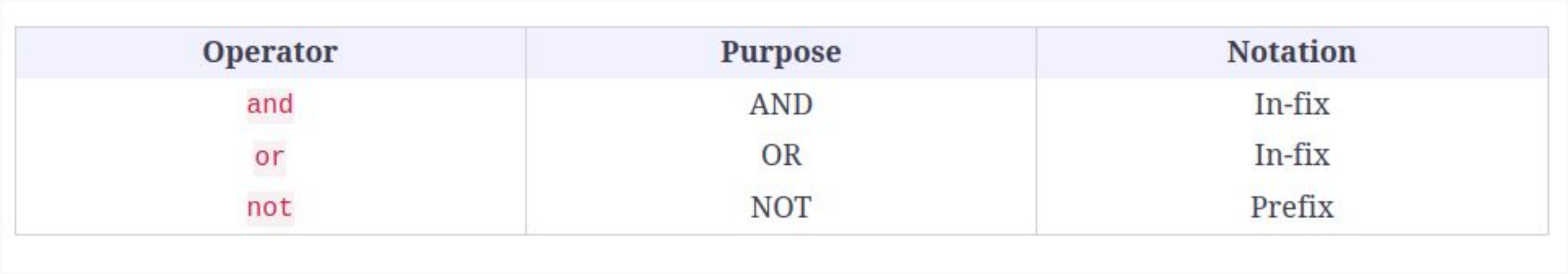


Operator	Purpose	Notation
=	Assign	In-fix
+=	Add and Assign	In-fix
-=	Subtract and Assign	In-fix
*=	Multiply and Assign	In-fix
/=	Divide and Assign	In-fix
//=	Divide, Floor, and Assign	In-fix
**=	Raise power and Assign	In-fix
%=	Take Modulo and Assign	In-fix
|=	OR and Assign	In-fix
&=	AND and Assign	In-fix
^=	XOR and Assign	In-fix
>>=	Right-shift and Assign	In-fix
<<=	Left-shift and Assign	In-fix

Demo Assignment Operators

Logical Operators

- Logical operators are used to manipulate the logic of Boolean expressions.



Operator	Purpose	Notation
`and`	AND	In-fix
`or`	OR	In-fix
`not`	NOT	Prefix

String Operations

Concatenation

- The “+” operator can be used to combine two strings together
- The “*” operator can be used to multiply and creating a repeating pattern

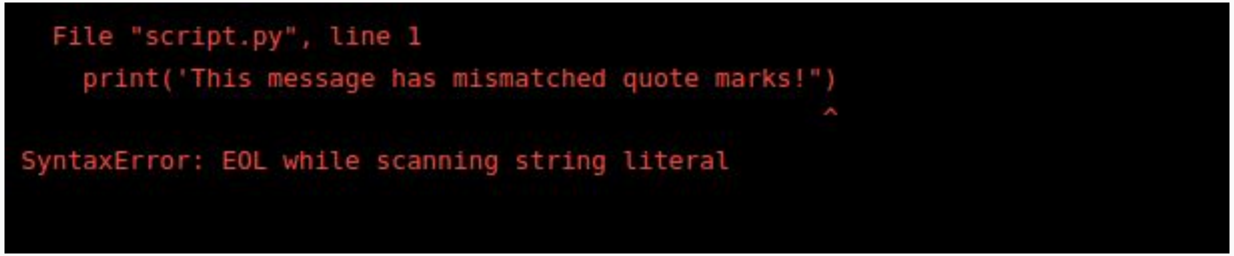
Demo Logical Operators

Errors

- Errors are ways that python attempts to understand and explain the mistakes made in programs by humans.
- The ^ character is used to point to where an error is made.
- These errors that we didn't expect to encounter can also be referred to as ***bugs.***
- The process of fixing the program to no longer produce unexpected errors is called ***debugging.***
- The 2 types of errors are Syntax Errors and Exceptions

Types of Errors - Syntax Errors

- This error is thrown when the syntax of the python programming language is violated
- It is also known as ***parsing*** error
- Remember "**syntax**" is structure - maybe missing parenthesis, punctuation, a command where not expected ...



```
  File "script.py", line 1
    print('This message has mismatched quote marks!")
                                                     ^
SyntaxError: EOL while scanning string literal
```

Types of Errors - Exceptions

- A syntactically correct statement can still cause an error when it is being executed
- Errors that are detected during execution are called **Exceptions**
- Most exceptions are not handled by the programs, so they result in the following errors:
- *Value Error, Name Error, Index Error, Module Not Found Error, Type Error, Zero Division Error*

Types of Errors - Exceptions

- **Value Error** - *thrown when a function's argument is of the correct type, but inappropriate value.*
- **Name Error** - *thrown when an object could not be found*
- **Index Error** - *thrown when trying to access an item at an invalid index*
- **Module Not Found Error** - *thrown when a module can not be found*
- **Type Error** - *thrown when a function's argument is of an inappropriate type*
- **Zero Division Error** - *thrown when the second operator in the division is zero*

Knowledge Check

What will be the output of the following piece of code?

```python
my_string = "0123456789"
print(my_string[-2: -6: -2])
```

A. 5432.

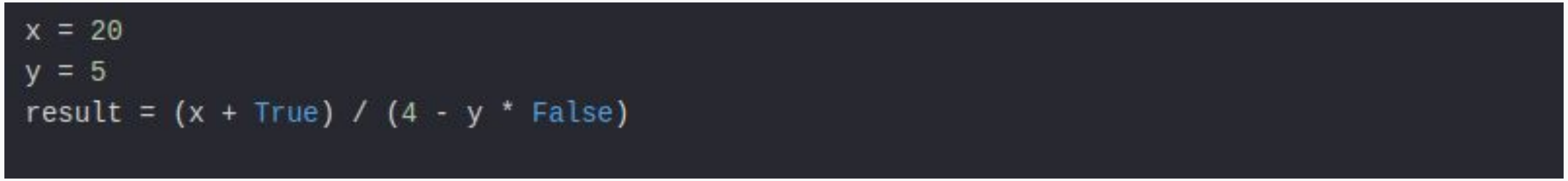
B. 8765.

C. 53.

[✓ D. 86.]

Knowledge Check

What is the value of result at the end of the following code?



```python
x = 20
y = 5
result = (x + True) / (4 - y * False)
```

A. False.

B. -21.

C. 5.

[✓ D. 5.25.]

Knowledge Check

A comparison operation always returns a value of the ______ data type.

[✓ A. Boolean.]

B. Integer.

C. Floating-Point Number.

D. String.

Knowledge Check

Which of these statements is true about operator precedence?

A. + has a higher precedence than -.

B. * has a lower precedence than -

[✓ C. * and / have the same precedence.]

D. None of the above.

Knowledge Check

String indices can be floats.

A. True

[✓ B. False]

Knowledge Check

modulo_variable = 14 % 4

A. 3.5

[✓ B. 2]

C. 56

Knowledge Check

How do you combine two strings?

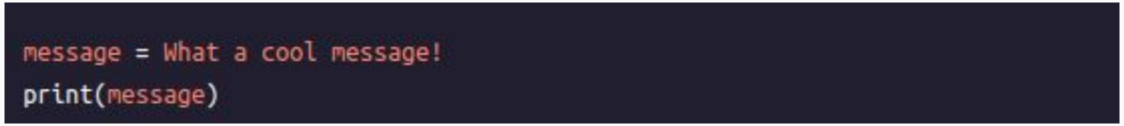
A. string1.concatenate(string2)

B. string1.combine(string2)

[✓ C. string1 + string2]

Knowledge Check

What happens when running the following code?



```
message = What a cool message!
print(message)
```

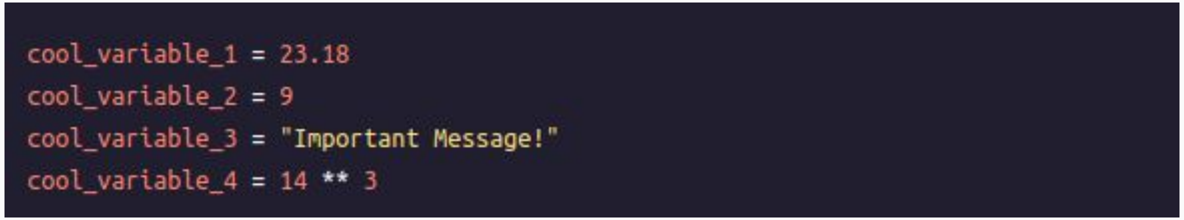
[✓ A. Python throws a SyntaxError because the string is not surrounded by quotes.]

B. “What a cool message!” gets printed to the console.

C. “message” is printed.

Knowledge Check

Which of the following defined variables is a string?



```
cool_variable_1 = 23.18
cool_variable_2 = 9
cool_variable_3 = "Important Message!"
cool_variable_4 = 14 ** 3
```

[✓ A. cool_variable_3]

B. cool_variable_1

C. cool_variable_4

D. cool_variable_2

Knowledge Check

What is the difference between a float and an int?

[✓ A. A **float** represents decimal quantities. An **int** represents whole numbers.]

B. A **float** can only be used with whole numbers. An **int** can be any number with a decimal point

C. A **float** contains text information. An **int** is any number.

Knowledge Check

What is the value of total_cost that gets printed?

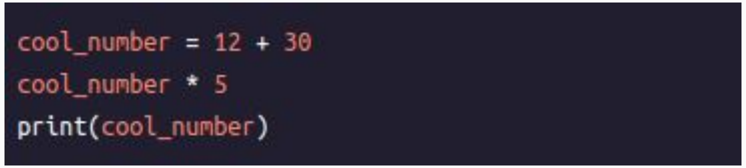
A. 10

[✓ B. 15]

C. 5

Knowledge Check

What is the output of the following code?



```
cool_number = 12 + 30
cool_number * 5
print(cool_number)
```

[✓ A. 42]

B. cool_number

C. 210

Knowledge Check

Which of the following will produce a SyntaxError?

A. answer = “Is this an error”

B. answer = is_this_an_error

[✓ C. answer = “Is this an error’]