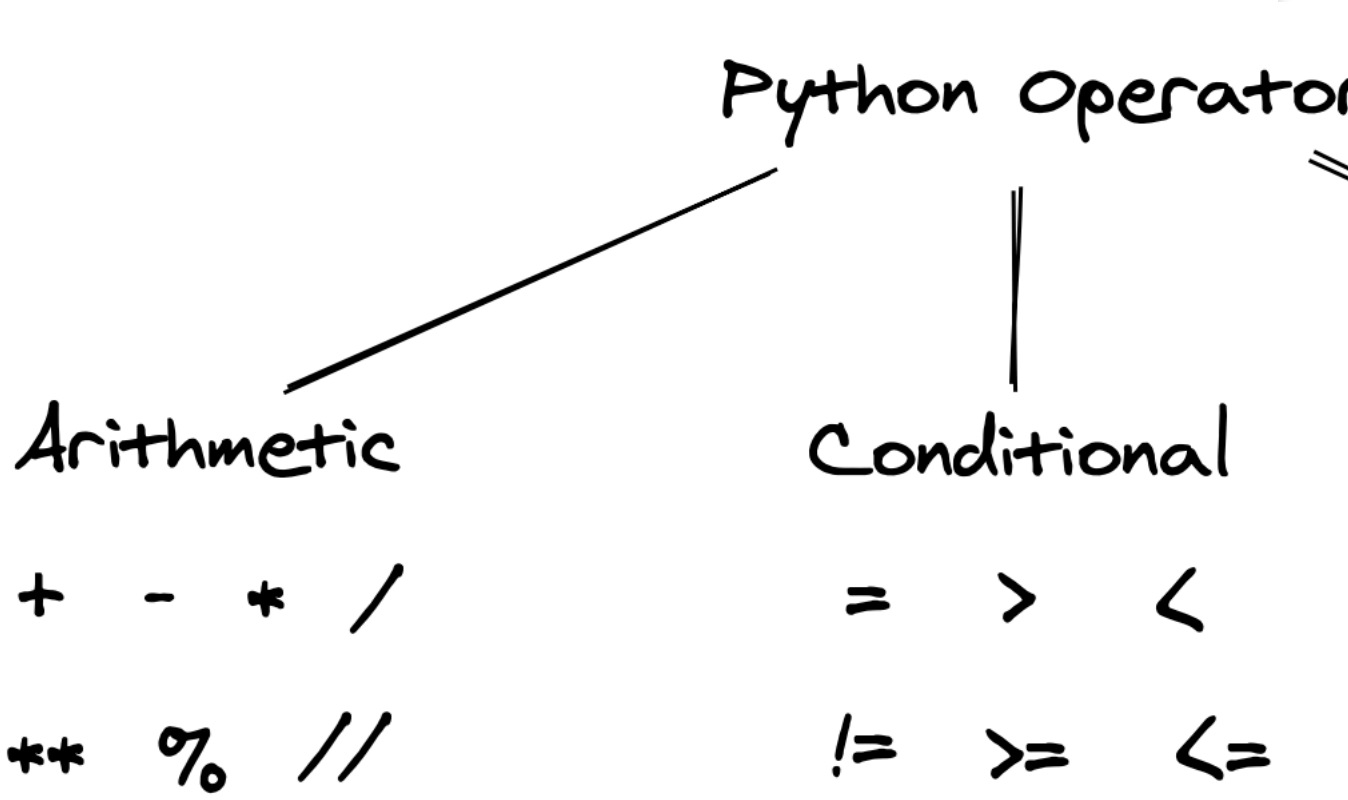
First Steps with Python and Jupyter

This tutorial is a part of Data Analysis with Python: Zero to Pandas and Zero to Data Analyst Science Bootcamp.



In [ ]:

In [ ]:

This tutorial covers the following topics:

- Performing arithmetic operations using Python
- Solving multi-step problems using variables
- Evaluating conditions using Python
- Combining conditions with logical operators
- Adding text styles using Markdown

Performing Arithmetic Operations using Python

Let's begin by using Python as a calculator. You can write and execute Python using a code cell within Jupyter.

> **Working with cells**: To create a new cell within Jupyter, you can select "Insert > Insert Cell Below" from the menu bar or just press the "+" button on the toolbar. You can also use the keyboard shortcut `Esc+B` to create a new cell. Once a cell is created, click on it to select it. You can then change the cell type to code or markdown (text) using the "Cell > Cell Type" menu option. You can also use the keyboard shortcuts `Esc+Y` and `Esc+M`. Double-click a cell to edit the content within the cell. To apply your changes and run a cell, use the "Cell > Run Cells" menu option or click the "Run" button on the toolbar or just use the keyboard shortcut `Shift+Enter`. You can see a full list of keyboard shortcuts using the "Help > Keyboard Shortcuts" menu option.

Run the code cells below to perform calculations and view their result. Try changing the numbers and run the modified cells again to see updated results. Can you guess what the `//`, `%`, and `**` operators are used for?

In [1]:
```
2+3
```

Out [1]: 5

In [ ]:

In [1]:
```
2 + 3 + 10
```

Out [1]: 15

In [2]:
```
3+4
```

Out [2]: 7

In [ ]:
```
99 - 73
```

Out [3]: 26

In [ ]:
```
23.54 * -1432
```

Out [3]: -33709.28

In [ ]:
```
100 / 7
```

Out [4]: 14.285714285714286

In [ ]:
```
100 // 7
```

Out [5]: 14

In [ ]:
```
100 % 7
```

Out [6]: 2

In [ ]:
```
5 ** 3
```

Out [7]: 125

As you might expect, operators like `/` and `*` take precedence over other operators like `+` and `-` as per mathematical conventions. You can use parentheses, i.e. `(` and `)`, to specify the order in which operations are performed.

In [ ]:
```
((2 + 5) * (17 - 3)) / (4 ** 3)
```

Out [8]: 1.53125

Python supports the following arithmetic operators:

Operator	Purpose	Example	Result
`+`	Addition	`2 + 3`	`5`
`-`	Subtraction	`3 - 2`	`1`
`*`	Multiplication	`8 * 12`	`96`
`/`	Division	`100 / 7`	`14.28..`
`//`	Floor Division	`100 // 7`	`14`
`%`	Modulus/Remainder	`100 % 7`	`2`
`**`	Exponent	`5 ** 3`	`125`

Try solving some simple problems from this page: https://www.math-only-math.com/worksheet-on-word-problems-on-four-operations.html .

You can use the empty cells below and add more cells if required.

In [ ]:
```

```

Solving multi-step problems using variables

Let's try solving the following word problem using Python:

> A grocery store sells a bag of ice for $1.25 and makes a 20% profit. If it sells 500 bags of ice, how much total profit does it make?

We can list out the information provided and gradually convert the word problem into a mathematical expression that can be evaluated using Python.

Cost of ice bag ($) = 1.25

Profit margin = 20% = .2

Profit per bag ($) = profit margin *cost of ice bag* = *.2* 1.25

No. of bags = 500

Total profit = no. of bags *profit per bag* = *500* (.2 * 1.25)

In [ ]:
```
500 * (.2 * 1.25)
```

Thus, the grocery store makes a total profit of $125. While this is a reasonable way to solve a problem, it's not entirely clear by looking at the code cell what the numbers represent. We can give names to each of the numbers by creating Python *variables*.

> **Variables**: While working with a programming language such as Python, information is stored in *variables*. You can think of variables as containers for storing data. The data stored within a variable is called its *value*.

In [ ]:
```
cost_of_ice_bag = 1.25
```

In [ ]:
```
profit_margin = .2
```

In [ ]:
```
number_of_bags = 500
```

The variables `cost_of_ice_bag`, `profit_margin`, and `number_of_bags` now contain the information provided in the word problem. We can check the value of a variable by typing its name into a cell. We can combine variables using arithmetic operations to create other variables.

> **Code completion**: While typing the name of an existing variable in a code cell within Jupyter, just type the first few characters and press the `Tab` key to autocomplete the variable's name. Try typing `pro` in a code cell below and press `Tab` to autocomplete to `profit_margin`.

In [ ]:
```
profit_margin
```

Out [8]: 0.2

In [ ]:
```
profit_per_bag = cost_of_ice_bag * profit_margin
```

In [ ]:
```
profit_per_bag
```

Out [15]: 0.25

In [ ]:
```
total_profit = number_of_bags * profit_per_bag
```

In [ ]:
```
total_profit
```

Out [17]: 125.0

If you try to view the value of a variable that has not been *defined*, i.e., given a value using the assignment statement `variable_name = value`, Python shows an error.

In [ ]:
```
net_profit
```

```
---------------------------------------------------------------------------NameError                                 Traceback (most recent call las
----> 1 net_profit
NameError: name 'net_profit' is not defined
```

Storing and manipulating data using appropriately named variables is a great way to explain what your code does.

Let's display the result of the word problem using a friendly message. We can do this using the `print` *function*.

> **Functions**: A function is a reusable set of instructions. It takes one or more inputs, performs certain operations, and often returns an output. Python provides many in-built functions like `print` and also allows us to define our own functions.

In [ ]:
```
print("The grocery store makes a total profit of $", total_profit)
```

```
The grocery store makes a total profit of $ 125.0
```

> **`print`**: The `print` function is used to display information. It takes one or more inputs, which can be text (within quotes, e.g., `"this is some text"`), numbers, variables, mathematical expressions, etc. We'll learn more about variables & functions in the next tutorial.

Creating a code cell for each variable or mathematical operation can get tedious. Fortunately, Jupyter allows you to write multiple lines of code within a single code cell. The result of the last line of code within the cell is displayed as the output.

Let's rewrite the solution to our word problem within a single cell.

In [ ]:
```
# Store input data in variables
cost_of_ice_bag = 1.25
profit_margin = .2
number_of_bags = 500

# Perform the required calculations
profit_per_bag = cost_of_ice_bag * profit_margin
total_profit = number_of_bags * profit_per_bag

# Display the result
print("The grocery store makes a total profit of $", total_profit)
```

```
The grocery store makes a total profit of $ 125.0
```

Note that we're using the `#` character to add *comments* within our code.

> **Comments**: Comments and blank lines are ignored during execution, but they are useful for providing information to humans (including yourself) about what the code does. Comments can be inline (at the end of some code), on a separate line, or even span multiple lines.

Inline and single-line comments start with `#`, whereas multi-line comments begin and end with three quotes, i.e. `"""`. Here are some examples of code comments:

In [ ]:
```
my_favorite_number = 1 # an inline comment
```

In [ ]:
```
# This comment gets its own line
my_least_favorite_number = 3
```

In [ ]:
```
"""This is a multi-line comment.
Write as little or as much as you'd like.

Comments are really helpful for people reading
your code, but try to keep them short & to-the-point.

Also, if you use good variable names, then your code is
often self explanatory, and you may not even need comments!
```

```
"""
a_neutral_number = 5
```

EXERCISE: A travel company wants to fly a plane to the Bahamas. Flying the plane costs 5000 dollars. So far, 29 people have signed up for the trip. If the company charges 200 dollars per ticket, what is the profit made by the company? Create variables for each numeric quantity and use appropriate arithmetic operations.

In [ ]:

In [ ]:

Evaluating conditions using Python

Apart from arithmetic operations, Python also provides several operations for comparing numbers & variables.

Operator	Description
`==`	Check if operands are equal
`!=`	Check if operands are not equal
`>`	Check if left operand is greater than right operand
`<`	Check if left operand is less than right operand
`>=`	Check if left operand is greater than or equal to right operand
`<=`	Check if left operand is less than or equal to right operand

The result of a comparison operation is either `True` or `False` (note the uppercase `T` and `F`). These are special keywords in Python. Let's try out some experiments with comparison operators.

In [ ]:
```
my_favorite_number = 1
my_least_favorite_number = 5
a_neutral_number = 3
```

In [ ]:
```
# Equality check - True
my_favorite_number == 1
```

Out [3]: True

In [ ]:
```
# Equality check - False
my_favorite_number == my_least_favorite_number
```

Out [4]: False

In [ ]:
```
# Not equal check - True
my_favorite_number != a_neutral_number
```

Out [5]: True

In [ ]:
```
# Not equal check - False
a_neutral_number != 3
```

Out [6]: False

In [ ]:
```
# Greater than check - True
my_least_favorite_number > a_neutral_number
```

Out [7]: True

In [ ]:
```
# Greater than check - False
my_favorite_number > my_least_favorite_number
```

Out [8]: False

In [ ]:
```
# Less than check - True
my_favorite_number < 10
```

Out [9]: True

In [ ]:
```
# Less than check - False
my_least_favorite_number < my_favorite_number
```

Out [10]: False

In [ ]:
```
# Greater than or equal check - True
my_favorite_number >= 1
```

Out [11]: True

In [ ]:
```
# Greater than or equal check - False
my_favorite_number >= 3
```

Out [12]: False

In [ ]:
```
# Less than or equal check - True
3 + 6 <= 9
```

Out [13]: True

In [ ]:
```
# Less than or equal check - False
my_favorite_number + a_neutral_number <= 3
```

Out [14]: False

Just like arithmetic operations, the result of a comparison operation can also be stored in a variable.

In [ ]:
```
cost_of_ice_bag = 1.25
is_ice_bag_expensive = cost_of_ice_bag >= 10
print("Is the ice bag expensive?", is_ice_bag_expensive)
```

```
Is the ice bag expensive? False
```

Combining conditions with logical operators

The logical operators `and`, `or` and `not` operate upon conditions and `True` & `False` values (also known as *booleans*). `and` and `or` operate on two conditions, whereas `not` operates on a single condition.

The `and` operator returns `True` when both the conditions evaluate to `True`. Otherwise, it returns `False`.

a	b	a and b
True	True	True
True	False	False
False	True	False
False	False	False

In [ ]:
```
my_favorite_number
```

Out [16]: 1

In [ ]:
```
my_favorite_number > 0 and my_favorite_number <= 3
```

Out [17]: True

In [ ]:
```
my_favorite_number < 0 and my_favorite_number <= 3
```

Out [18]: False

In [ ]:
```
my_favorite_number > 0 and my_favorite_number >= 3
```

Out [19]: False

In [ ]:
```
True and False
```

Out [20]: False

In [ ]:
```
True and True
```

Out [21]: True

The `or` operator returns `True` if at least one of the conditions evaluates to `True`. It returns `False` only if both conditions are `False`.

a	b	a or b
True	True	True
True	False	True
False	True	True
False	False	False

In [ ]:
```
a_neutral_number = 3
```

In [ ]:
```
a_neutral_number == 3 or my_favorite_number < 0
```

Out [23]: True

In [ ]:
```
a_neutral_number != 3 or my_favorite_number < 0
```

Out [24]: False

In [ ]:
```
my_favorite_number < 0 or True
```

Out [25]: True

In [ ]:
```
False or False
```

Out [26]: False

The `not` operator returns `False` if a condition is `True` and `True` if the condition is `False`.

In [ ]:
```
not a_neutral_number == 3
```

Out [27]: False

In [ ]:
```
not my_favorite_number < 0
```

Out [28]: True

In [ ]:
```
not False
```

Out [29]: True

In [ ]:
```
not True
```

Out [30]: False

Logical operators can be combined to form complex conditions. Use round brackets or parentheses `(` and `)` to indicate the order in which logical operators should be applied.

In [ ]:
```
(2 > 3 and 4 <= 5) or not (my_favorite_number < 0 and True)
```

Out [31]: True

In [ ]:
```
not (True and 0 < 1) or (False and True)
```

Out [32]: False

If parentheses are not used, logical operators are applied from left to right.

In [ ]:
```
not True and 0 < 1 or False and True
```

Out [33]: False

Experiment with arithmetic, conditional and logical operators in Python using the interactive nature of Jupyter notebooks. We will learn more about variables and functions in future tutorials.

Adding text styles using Markdown

Adding explanations using text cells (like this one) is a great way to make your notebook informative for other readers. It is also useful if you need to refer back to it in the future. You can double click on a text cell within Jupyter to edit it. In the edit mode, you'll notice that the text looks slightly different (for instance, the heading has a `##` prefix. This text is written using Markdown, a simple way to add styles to your text. Execute this cell to see the output without the special characters. You can switch back and forth between the source and the output to apply a specific style.

For instance, you can use one or more `#` characters at the start of a line to create headers of different sizes:

Header 1

Header 2

Header 3

Header 4

To create a bulleted or numbered list, simply start a line with `*` or `1.`.

A bulleted list:

- Item 1
- Item 2
- Item 3

A numbered list:

1. Apple
2. Banana
3. Pineapple

You can make some text bold using `**`, e.g., **some bold text**, or make it italic using `*`, e.g., *some italic text.* You can also create links, e.g., a link. Images are easily embedded too:

Another really nice feature of Markdown is the ability to include blocks of code. Note that code blocks inside Markdown cells cannot be executed.

```
# Perform the required calculations
profit_per_bag = cost_of_ice_bag * profit_margin
total_profit = number_of_bags * profit_per_bag

# Display the result
print("The grocery store makes a total profit of $", total_profit)
```

You can learn the full syntax of Markdown here: https://learnxinyminutes.com/docs/markdown/

In [ ]:
```
2**3*2
```

Out [1]: 16

Questions for Revision

Try answering the following questions to test your understanding of the topics covered in this notebook:

1. What is a Jupyter notebook?
2. How do you add a new code cell below an existing cell?
3. How do you add a new Markdown cell below an existing cell?
4. How do you convert a code cell to a Markdown cell or vice versa?
5. How do you execute a code cell within Jupyter?
6. What the different arithmetic operations supported in Python?
7. How do you perform arithmetic operations using Python?
8. What is the difference between the `/` and the `//` operators?
9. What is the difference between the `*` and the `**` operators?
10. What is the order of precedence for arithmetic operators in Python?
11. How do you specify the order in which arithmetic operations are performed in an expression involving multiple operators?
12. How do you solve a multi-step arithmetic word problem using Python?
13. What are variables? Why are they useful?
14. How do you create a variable in Python?
15. What is the assignment operator in Python?
16. What are the rules for naming a variable in Python?
17. How do you view the value of a variable?
18. How do you store the result of an arithmetic expression in a variable?
19. What happens if you try to access a variable that has not been defined?
20. How do you display messages in Python?
21. What type of inputs can the print function accept?
22. What are code comments? How are they useful?
23. What are the different ways of creating comments in Python code?
24. What are the different comparison operations supported in Python?
25. What is the result of a comparison operation?
26. What is the difference between `=` and `==` in Python?
27. What are the logical operators supported in Python?
28. What is the difference between the `and` and `or` operators?
29. Can you use comparison and logical operators in the same expression?
30. What is the purpose of using parentheses in arithmetic or logical expressions?
31. What is Markdown? Why is it useful?
32. How do you create headings of different sizes using Markdown?
33. How do you create bulleted and numbered lists using Markdown?
34. How do you create bold or italic text using Markdown?
35. How do you include links & images within Markdown cells?
36. How do you include code blocks within Markdown cells?
37. Is it possible to execute the code blocks within Markdown cells?
38. How do you upload and share your Jupyter notebook online using Jovian?
39. What is the purpose of the API key requested by jovian.commit ? Where can you find the API key?
40. Where can you learn about arithmetic, conditional and logical operations in Python?