Introduction to Programming with Python

Installing Python and setting up the environment (Anaconda, Jupyter Notebook)

How to run the code

This tutorial is an executable Jupyter notebook (don't worry if these terms seem unfamiliar; we'll learn more about them soon). You can *run* this tutorial and experiment with the code examples in a couple of ways: *using free online resources* (recommended) or *on your computer*.

Option 1: Running using free online resources (1-click, recommended)

The easiest way to start executing the code is to click the **Run** button at the top of this page and select **Run on Binder**. You can also select "Run on Colab" or "Run on Kaggle", but you'll need to create an account on Google Colab or Kaggle to use these platforms.

Option 2: Running on your computer locally

To run the code on your computer locally, you'll need to set up Python, download the notebook and install the required libraries. We recommend using the Conda distribution of Python. Click the **Run** button at the top of this page, select the **Run Locally** option, and follow the instructions.

> **Jupyter Notebooks**: This tutorial is a Jupyter notebook - a document made of *cells*. Each cell can contain code written in Python or explanations in plain English. You can execute code cells and view the results, e.g., numbers, messages, graphs, tables, files, etc. instantly within the notebook. Jupyter is a powerful platform for experimentation and analysis. Don't be afraid to mess around with the code & break things - you'll learn a lot by encountering and fixing errors. You can use the "Kernel > Restart & Clear Output" menu option to clear all outputs and start again from the top.

```
Start coding or generate with AI.
```

Basic syntax, data types, and operations

Print Function

Use the `print()` function to display output.

```
print('Hello')
```

```
Hello
```

```
print("This is my first programming class")
```

```
This is my first programming class
```

```
print('522222')
```

```
522222
```

```
print('Hello world')
```

```
Hello world
```

```
Start coding or generate with AI.
```

```
print("Hello", 3, 'Mat')
```

```
Hello 3 Mat
```

```
print("I'm a good student")
```

```
I'm a good student
```

```
print('a', 'b', 'c', 'd', sep=', ')
```

```
a, b, c, d
```

```
print('a', 'b', 'c', 'd', sep='-')
```

```
a-b-c-d
```

```
print('a, b, c, d')
```

```
a, b, c, d
```

Comments

Python has commenting capability for in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment.

```
# Addition

2 + 3
```

```
5
```

```
x = 3
```

```
X = 4
```

```
x + 3
```

```
6
```

```
X + 3
```

```
7
```

```
Start coding or generate with AI.
```

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`

Addition

```
2 + 3+ 5 + 6
```

```
16
```

```
Start coding or generate with AI.
```

Subtraction

```
14 -12
```

```
2
```

```
14 - 29
```

```
-15
```

```
Start coding or generate with AI.
```

Multiplication

```
2 * 3
```

```
6
```

```
14 * 4
```

```
56
```

Division

```
14 / 2
```

```
7.0
```

```
15/2
```

```
7.5
```

```
Start coding or generate with AI.
```

Floor Division

```
15//2
```

```
7
```

```
14//2
```

```
7
```

Modulus/Remainder

```
15 % 2
```

```
1
```

```
14 % 2
```

```
0
```

```
14 % 3
```

```
2
```

Exponential

```
2 ** 3
```

```
8
```

```
2 * 2 * 2
```

```
8
```

```
Start coding or generate with AI.
```

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

```
Start coding or generate with AI.
```

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.

```
2 + 3 * 5
```

```
17
```

```
(3 + 5) * 4
```

```
32
```

```
Start coding or generate with AI.
```

$x^2 + 3x + 5$

x^2 + 3x + 5

$$\frac{2}{2x}$$

In programming, a keyword is a word that is reserved by a program because the word has a special meaning. Keywords can be commands or parameters. Every programming language has a set of keywords that `cannot be used as variable names`

```
x = 3
```

```
x + 5
```

```
8
```

```
x * 20
```

```
60
```

```
Start coding or generate with AI.
```

Rules for setting Identifiers

- can only start with an alphabet or _
- Followed by 0 or more letter,_ and digits
- keywords cannot be used as an identifiers

```
Start coding or generate with AI.
```

Variable

A variable is created using an assignment statement. It begins with the variable's name, followed by the assignment operator `=` followed by the value to be stored within the variable. Note that the assignment operator `=` is different from the equality comparison operator `==`.

```
name = "titilayo"
```

```
1name = "titilayo"
```

```
  Cell In[88], line 1
    1name = "titilayo"
    ^
SyntaxError: invalid decimal literal
```

```
my-name = "titilayo"
```

```
  Cell In[90], line 1
    my-name = "titilayo"
    ^
SyntaxError: cannot assign to expression here. Maybe you meant '==' instead of '='?
```

```
my_name = "titilayo"
```

```
name2 = "Peace"
```

```
print(name2)
```

```
Peace
```

```
and = "titilayo"
```

```
  Cell In[98], line 1
    and = "titilayo"
    ^
SyntaxError: invalid syntax
```

```
import keyword
```

```
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'fo
```

```
print(len(keyword.kwlist))
```

```
35
```

```
lambda = "my name is Titi"
```

```
  Cell In[1], line 1
    lambda = "my name is Titi"
           ^
SyntaxError: invalid syntax
```

```
Lambda = "my name is Titi"
```

```
x, y, z = 2, 3, 4
```

```
x
```

```
2
```

```
x = y = z = 4
```

```
y
```

```
4
```

```
fruits
```

```
{'apple', 'banana', 'cherry'}
```

```
best_fruits = ['apple', 'banana', 'cherry']
```

```
x, y, z = best_fruits
```

```
x
```

```
'apple'
```

Data Types

Any data or information stored within a Python variable has a type. You can view the type of data stored within a variable using the type function.

```
type(2)
```

```
int
```

```
type(2.0)
```

```
float
```

```
name = 'Matthew'
```

```
type(name)
```

```
str
```

```
number = 123456789
```

```
type(number)
```

```
int
```

```
type(True)
```

```
bool
```

```
type(False)
```

```
bool
```

```
fruits = ['apple', 'banana', 'cherry']
```

```
type(fruits)
```

```
list
```

```
fruits = ('apple', 'banana', 'cherry')
```

```
type(fruits)
```

```
tuple
```

```
fruits = {'apple', 'banana', 'cherry'}
```

```
type(fruits)
```

```
set
```

Python has several built-in data types for storing different kinds of information in variables. Following are some commonly used data types:

1. Integer
2. Float
3. Boolean
4. None
5. String
6. List
7. Tuple
8. Dictionary

Integer, float, boolean, None, and string are *primitive data types* because they represent a single value. Other data types like list, tuple, and dictionary are often called *data structures* or *containers* because they hold multiple pieces of data together.

- Integer

Integers represent positive or negative whole numbers, from negative infinity to infinity. Note that integers should not include decimal points. Integers have the type `int`.

```
type(-1234)
```

```
int
```

```
num = -1234.
```

```
type(num)
```

```
float
```

```
Start coding or generate with AI.
```

Unlike some other programming languages, integers in Python can be arbitrarily large (or small). There's no lowest or highest value for integers, and there's just one int type (as opposed to short, int, long, long long, unsigned int, etc. in C/C++/Java).

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

- Float

Floats (or floating-point numbers) are numbers with a decimal point. There are no limits on the value or the number of digits before or after the decimal point. Floating-point numbers have the type `float`.

```
x = 11 * 3.0
```

```
type(x)
```

```
float
```

```
Start coding or generate with AI.
```

Note that a whole number is treated as a float if written with a decimal point, even though the decimal portion of the number is zero.

```
type(3.0)
```

```
float
```

Floating point numbers can also be written using the scientific notation with an "e" to indicate the power of 10.

```
1e3
```

```
1000.0
```

```
1e2
```

```
100.0
```

```
Start coding or generate with AI.
```

You can convert floats into integers and vice versa using the `float` and `int` functions. The operation of converting one type of value into another is called casting.

```
x
```

```
33.0
```

```
y = int(x)
```

```
y
```

```
33
```

```
Start coding or generate with AI.
```

While performing arithmetic operations, integers are automatically converted to `float`s if any of the operands is a `float`. Also, the division operator `/` always returns a `float`, even if both operands are integers. Use the `//` operator if you want the result of the division to be an `int`.

```
8/2
```

```
4.0
```

```
9/2
```

```
4.5
```

```
9//2
```

```
4
```

Complex

```
x = 2 + 3j
```

```
type(x)
```

```
complex
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

- ### Boolean

Booleans represent one of 2 values: `True` and `False`. Booleans have the type `bool`.

```
type(False)
```

```
bool
```

```
Start coding or generate with AI.
```

```
true = 'I am a boy'
```

```
type(true)
```

```
str
```

```
type(True)
```

```
bool
```

Booleans are generally the result of a comparison operation, e.g., `==`, `>=`, etc.

```
5 == 2+6
```

```
False
```

```
Start coding or generate with AI.
```

Booleans are automatically converted to `int`s when used in arithmetic operations. `True` is converted to `1` and `False` is converted to `0`.

Any value in Python can be converted to a Boolean using the `bool` function.

Only the following values evaluate to `False` (they are often called *falsy* values):

1. The value `False` itself
2. The integer `0`
3. The float `0.0`
4. The empty value `None`
5. The empty text `""`
6. The empty list `[]`
7. The empty tuple `()`
8. The empty dictionary `{}`
9. The empty set `set()`
10. The empty range `range(0)`

Everything else evaluates to `True` (a value that evaluates to `True` is often called a *truthy* value).

```
Start coding or generate with AI.
```

- String

A string is used to represent text (*a string of characters*) in Python. Strings must be surrounded using quotations (either the single quote `'` or the double quote `"`). Strings have the type `string`.

```
a = "Hello World!"
```

```
a
```

```
'Hello World!'
```

```
type(a)
```

```
str
```

```
int(a)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[87], line 1
----> 1 int(a)

ValueError: invalid literal for int() with base 10: 'Hello World!'
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Strings created using single or double quotes must begin and end on the same line. To create multiline strings, use three single quotes `'''` or three double quotes `"""` to begin and end the string. Line breaks are represented using the newline character `\n`.

```
phrase = 'I am a good boy, i love reading'
```

```
phrase
```

```
'I am a good boy, i love reading'
```

```
phrase = '''I am a good boy,
i love reading'''
```

```
phrase
```

```
'I am a good boy,\ni love reading'
```

```
phrase2 = """Son: 'Dad, i like you.'
Dad: 'Thank you' """
```

```
phrase2
```

```
"Son: 'Dad, i like you.'\nDad: 'Thank you' "
```

```
Start coding or generate with AI.
```

Check

```
text = "The most important things in life are free"
```

```
print('Free' in text)
```

```
False
```

```
print('air' not in text)
```

```
True
```

```
Start coding or generate with AI.
```

Slicing

```
a
```

```
'Hello World!'
```

```
a[1:6]
```

```
'ello '
```

```
a[0:7]
```

```
'Hello W'
```

Case

```
a
```

```
'Hello World!'
```

```
a.upper()
```

```
'HELLO WORLD!'
```

```
a = a.lower()
```

```
a
```

```
'hello world!'
```

strip

```
a = '     Hello World      '
```

```
a
```

```
'     Hello World      '
```

```
a.strip()
```

```
'Hello World'
```

Concatenate

```
a = 'Hello'
b= 'World'
```

```
c = a+b
```

```
c
```

```
'HelloWorld'
```

```
d = a + "-" + b
```

```
d
```

```
'Hello-World'
```

```
age = 36
```

```
print("My name is John, I am", age)
```

```
My name is John, I am 36
```

F - String

```
print(f'I am {age} years old')
```

```
I am 36 years old
```

```
print(f'I am  {age} years old')
```

```
I am  29 years old
```

```
name = "John"
age = 29
price = 59.345
```

```
details = {
    'name': 'Matthew',
    'age': 29,
    'salary_expectation': '$15k'
}
```

```
details
```

```
{'name': 'Matthew', 'age': 29, 'salary_expectation': '$15k'}
```

```
type(details)
```

```
dict
```

Start coding or generate with AI.

```
print(f'My name is {name}, I am {age} years old.')
```

```
My name is John, I am 29 years old.
```

```
print(f'The price of the goat is ${price}')
```

```
The price of the goat is $59.345
```

```
print(f'The price of the goat is ${price:.2f}')
```

```
The price of the goat is $59.34
```

```
x = 1500.867
```

```
import math
```

```
math.ceil(x)
```

```
1501
```

```
Start coding or generate with AI.
```

Random Number

```
import random
```

```
random.randint(1, 10000)
```

```
10000
```

```
random.randrange(1, 10)
```

```
6
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```