Introduction To Python Programming

Introduction to Python programming Course Outline

- Intro to Computer Science
- Environment Setup (Anaconda)
- Command Line
- Conda & pip package managers
- Jupyter Notebook
- Input & Output
- Variables
- **Data types**
- Numbers & Math
- Lists
- Tuples
- Sets
- Dictionaries
- Advanced If and Loops
- List Comprehensions
- Dictionary Comprehensions
- Exceptions
- File Handling

Data types

Name	Type	Description
Integers	int	Whole numbers, such as: **3 300 200**
Floating point	float	Numbers with a decimal point: **2.3 4.6 100.0**
Strings	str	Ordered sequence of characters: **"hello" 'Sammy' "2000" "楽しい"**
Lists	list	Ordered sequence of objects: **[10,"hello",200.3]**
Dictionaries	dict	Unordered Key:Value pairs: **{"mykey" : "value" , "name" : "Frankie"}**
Tuples	tup	Ordered immutable sequence of objects: **(10,"hello",200.3)**
Sets	set	Unordered collection of unique objects: **{"a","b"}**
Booleans	bool	Logical value indicating **True** or **False**

Data types

Python has a lot of built-in data types
Each variable has a data type based on the value assigned to it
You can check a variable's type using type() function

```python
# use type( ) function to check what is the type of a variable
x = 5
y = "python is awesome"
z = [1, 2, 3]

print(type(x)) # int
print(type(y)) # str
print(type(z)) # list
```

Introduction to Python programming Course Outline

- Intro to Computer Science
- Environment Setup (Anaconda)
- Command Line
- Conda & pip package managers
- Jupyter Notebook
- Input & Output
- Variables
- Data types
- **Numbers & Math**
- Lists
- Tuples
- Sets
- Dictionaries
- Advanced If and Loops
- List Comprehensions
- Dictionary Comprehensions
- Exceptions
- File Handling

Numbers & Math

There are two types of numbers in Python: integers (int) and floating point (float)

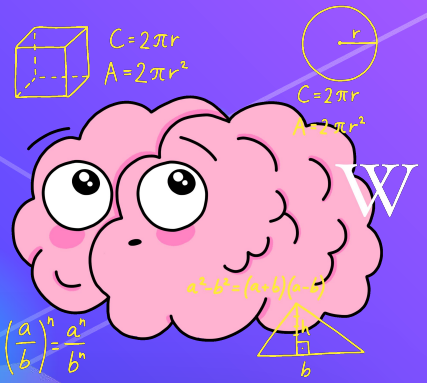
Scientific notation is used to describe very large numbers
2.5e2 = 2.5 x 10 ^ 2 = 250

```
# this is integer values (int)
5
1000
-5000
0


# this is float values (float)
5.25
1000.75
-5000.3
5.0
2.5e2   # 2.5*(10**2)
2.5e+2  # 2.5*(10**2)
2.5e-2  # 2.5*(10**-2)
```

Numbers & **Math**

Python supports a number of arithmetic operations

Operator	Description	Syntax
+	Addition: adds two operands	x + y
-	Subtraction: subtracts two operands	x - y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when first operand is divided by the second	x % y
**	Power : Returns first raised to power second	x ** y




```
3 + 5        # result is 8
10 - 7       # result is 3
2 * 5        # result is 10
15 / 5       # result is 3
3 / 2        # result is 1.5
3 // 2       # result is 1
32 % 3       # result is 2
2 ** 3       # result is 8
4 ** 0.5     # result is 2
```

Numbers & **Math**

You can combine an operator with the assignment expression (=) to update a variable's value

For example, '+=' increments the variable on the left hand side by the value on the right hand side

And '*=' multiplies the variable on the left hand side by the value on the right hand side

```
x += 5   # x = x + 5
x -= 5   # x = x - 5
x *= 5   # x = x * 5
x /= 5   # x = x / 5
x %= 5   # x = x % 5
x //= 5 # x = x // 5
x **= 5 # x = x ** 5
```

Quiz Time!

Q1. 3 + 3 * 3 + 3

- A. 36
- B. 15
- C. 27

Q2. (3 + 3) * (3 + 3)

- A. 36
- B. 15
- C. 27

Q3. 4 // 2 + 5 * (1+2)

- A. 21
- B. 9
- C. 17

Q4. 3 + 3 / 3 - 3

- A. 1
- B. ZeroDivisionError
- C. 0

Q5. (3 + 3) / (3 – 3)

- A. 1
- B. ZeroDivisionError
- C. 0

Introduction to Python programming Course Outline

- Intro to Computer Science
- Environment Setup (Anaconda)
- Command Line
- Conda & pip package managers
- Jupyter Notebook
- Input & Output
- Variables
- Data types
- Numbers & Math
- Lists
- Tuples
- Sets
- Dictionaries
- Advanced If and Loops
- List Comprehensions
- Dictionary Comprehensions
- Exceptions
- File Handling

Boolean & Comparison and Logic

Boolean algebra is the type of algebra performed on Boolean values only. Those are, True and False (0 and 1)

```
is_online = True

has_dog = False
```

Boolean & **Comparison** and Logic

Comparisons yield a Boolean value
(Assume a = 10 & b = 20)

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

```
5 == 5      # result is True
5 != 5      # result is False
10 > 7      # result is True
2 >= 5      # result is False
15 < 5      # result is False
3 <= 3      # result is True
```

Boolean & Comparison and **Logic**

OPERATOR	DESCRIPTION	SYNTAX
and	Like multiplication: 1 x 0 = 0 Logical AND: True if both the operands are true	x and y
or	Like addition: 1 + 0 = 1 Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if operand is false	not x

```python
# AND
1 < 2 and 2 < 3          # Result is True
1 != 1 and 2 < 3         # Result is False
1 != 1 and 2 > 3         # Result is False


# OR
1 < 2 or 2 < 3           # Result is True
1 != 1 or 2 < 3          # Result is True
1 != 1 or 2 > 3          # Result is False


# NOT
not 1 == 1               # Result is False
not 1 > 10               # Result is True
```

Quiz Time!

Q1. 5 > 10 and 3 > 2

- A. True
- B. False

Q2. 5 > 10 or 3 > 2

- A. True
- B. False

Q3. -10 < 3 and 0 < 2

- A. True
- B. False

Q4. (not 1 == 10) and 2 >= 2

- A. True
- B. False

Q5. 0 > -1 and (1 == 2 and (not 1 != 2))

- A. True
- B. False

Introduction to Python programming Course Outline

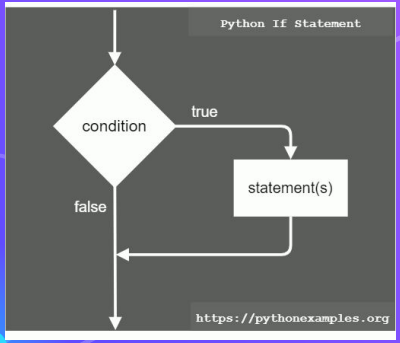
- Intro to Computer Science
- Environment Setup (Anaconda)
- Command Line
- Conda & pip package managers
- Jupyter Notebook
- Input & Output
- Variables
- Data types
- Numbers & Math
- Lists
- Tuples
- Sets
- Dictionaries
- Advanced If and Loops
- List Comprehensions
- Dictionary Comprehensions
- Exceptions
- File Handling

If Conditions

Previously, when we run our code, it would execute all statements in order

It's time to apply flow control

If statements allow us to control the flow of the code based on a certain condition




```python
person = 'George'

if person == 'Sammy':
    print('Welcome Sammy!')
elif person =='George':
    print('Welcome George!')
else:
    print("Welcome, what's your name?")

# Welcome George!
```

Quiz Time!

What will be the output of the following if statements:

Q1.

```
number1 = 5
number2 = 1
if (number1 + number2) < 3:
  print("Sloths")
else:
  print("Cats")
```

Q2.

```
num = 15
if num / 7 == 7:
  print("It divides by 7")
elif num / 3 == 5:
  print("It divides by 3")
else:
  print("Doesn't divide")
```

- A. Sloths
- B. Cats
- C. No print

- A. It divides by 7
- B. It divides by 3
- C. Doesn’t divide

Strings

Strings are ordered sequences of characters (alphabets, numbers, etc.)
Individual characters can be accessed using indexing

```
greeting = "Hello World"
greeting = 'Hello World'

print(greeting[0])  # H
print(greeting[2])  # l
print(greeting[-1]) # d
```

String Formatting

A way to inject a variable into a string for convenience

Add an ‘f’ before the string to add formatting,
then add variables using braces {}

```
x = 10
y = x / 2

print(f"Value of x = {x} and value of y = {y}")
# Value of x = 10 and value of y = 5.0
```

Introduction to Python programming Course Outline

- Intro to Computer Science
- Environment Setup (Anaconda)
- Command Line
- Conda & pip package managers
- Jupyter Notebook
- Input & Output
- Variables
- Data types
- Numbers & Math
- **Lists**
- Tuples
- Sets
- Dictionaries
- Advanced If and Loops
- List Comprehensions
- Dictionary Comprehensions
- Exceptions
- File Handling

Lists

Lists are the most common data structure in Python

You can store multiple values (elements) inside a single variable

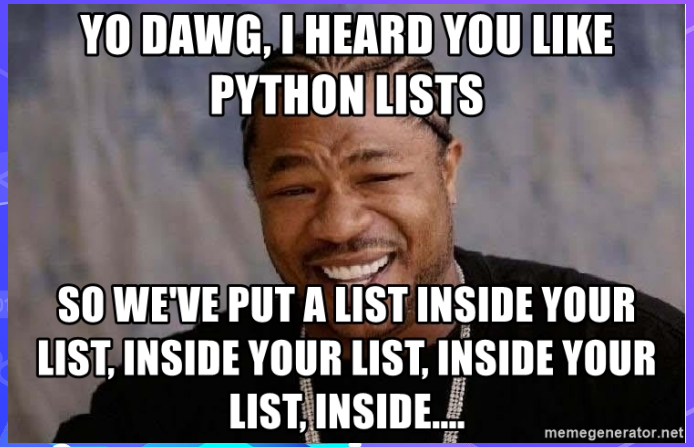
Unlike other programming languages, Python lists can have elements of different types

```python
my_list = ['A string', 23, 100.232 , 'p', True]

print(my_list[0])   # 'A string'
print(my_list[1])   # 23
print(my_list[2])   # 100.232
print(my_list[3])   # 'p'
print(my_list[4])   # True
```

Lists

List elements can be lists too!

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

print(my_list[0])        # [1,2,3]
print(my_list[0][1])     # 2
print(my_list[2][1][1]) # 9
```



Introduction to Python programming Course Outline

- Intro to Computer Science
- Environment Setup (Anaconda)
- Command Line
- Conda & pip package managers
- Jupyter Notebook
- Input & Output
- Variables
- Data types
- Numbers & Math
- Lists
- **Tuples**
- Sets
- Dictionaries
- Advanced If and Loops
- List Comprehensions
- Dictionary Comprehensions
- Exceptions
- File Handling

Tuples (faster and immutable lists)

Used when you have immutable values and need faster processing on them

```python
my_tuple = ('A string', 23, 100.232 , 'p', True)

print(my_tuple[0])  # 'A string'
print(my_tuple[1])  # 23
print(my_tuple[2])  # 100.232
print(my_tuple[3])  # 'p'
print(my_tuple[4])  # True
```

Questions ?!

Thanks!

>_ Live long and prosper