Python Basics Review

What is Python?

- **Introduction**: Python is a general-purpose programming language known for its simplicity and ease of use. Python is used in many fields like data science and machine learning, web development, scripting and automation, embedded systems and IoT, and much more.
- **Common Use Cases**: Python is used in data science, machine learning, web development, cybersecurity, automation and microcomputers like the Raspberry Pi and MicroPython-compatible boards.

Python in Your Local Environment

- **Installation**: The best way to install Python on Windows, Mac, and Linux is to download the installer from the official Python website (`https://www.python.org/`).

Variables

- **Declaring Variables**: To declare a variable, you start with the variable name followed by the assignment operator (`=`) and then the value. This can be a number, string, boolean, etc. Here are some examples:

▼ Example Code

```python
name = 'John Doe'
age = 25
```

- **Naming Conventions for Variables**: Here are the naming conventions you should use for variables:
  - Variable names can only start with a letter or an underscore (_), not a number.
  - Variable names can only contain alphanumeric characters (a-z, A-Z, 0-9) and underscores (_).
  - Variable names are case-sensitive — `age`, `Age`, and `AGE` are all considered unique.
  - Variable names cannot be one of Python's reserved keywords such as `if`, `class`, or `def`.
  - Variables names with multiple words are separated by underscores. Ex. `snake_case`.

Comments

- **Single Line Comments**: These types of comments should be used for short notes you wish to leave in your code.

▼ Example Code

```python
# This is a single line comment
```

- **Multi-line Strings**: These types of strings can be used to leave larger notes or to comment out sections of code.

▼ Example Code

```python
"""
This is a multi-line string.
Here is some code commented out.

name = 'John Doe'
age = 25
"""
```

- **`print()` Function**: To print data to the console, you can use the `print()` function like this:

▼ Example Code

```python
print('Hello world!') # Hello world!
```

Saved! Your code was saved to your browser's local storage.

variables. The language knows what the type of a variable is based on what you assign to the variable.

- **Integer**: A whole number without decimals:

▼ Example Code

```
my_integer_var = 10
print('Integer:', my_integer_var) # Integer: 10
```

- **Float**: A number with decimals:

▼ Example Code

```
my_float_var = 4.50
print('Float:', my_float_var) # Float: 4.5
```

- **String**: A sequence of characters wrapped in quotes:

▼ Example Code

```
my_string_var = 'hello'
print('String:', my_string_var) # String: hello
```

- **Boolean**: A value representing either `True` or `False`:

▼ Example Code

```
my_boolean_var = True
print('Boolean:', my_boolean_var) # Boolean: True
```

- **Set**: An unordered collection of unique elements:

▼ Example Code

```
my_set_var = {7, 5, 8}
print('Set:', my_set_var) # Set: {7, 5, 8}
```

- **Dictionary**: A collection of key-value pairs, enclosed in curly braces:

▼ Example Code

```
my_dictionary_var = {"name": "Alice", "age": 25}
print('Dictionary:', my_dictionary_var) # Dictionary: {'name': 'Alice', 'age': 25}
```

- **Tuple**: An immutable ordered collection, enclosed in parentheses:

▼ Example Code

```
my_tuple_var = (7, 5, 8)
print('Tuple:', my_tuple_var) # Tuple: (7, 5, 8)
```

- **Range**: A sequence of numbers, often used in loops:

▼ Example Code

```
my_range_var = range(5)
print(my_range_var) # range(0, 5)
```

- **List**: An ordered collection of elements that supports different data types:

▼ Example Code

- **None**: A special value that represents the absence of a value:

▼ Example Code

```
my_none_var = None
print('None:', my_none_var) # None: None
```

Immutable and Mutable Types

- **Immutable Types**: These types cannot change once declared, although you can point their variables at something new, which is called reassignment. They include integer, float, complex, boolean, string, tuple, range, and `None`.
- **Mutable Types**: These types can change once declared. You can add, remove, or update their items. They include collection types such as list, set, and dictionary.
- **`type()` Function**: To see the type for a variable, you can use the `type()` function like this:

▼ Example Code

```
greeting = 'Hello there!'
age = 21

print(type(greeting)) # <class 'str'>
print(type(age)) # <class 'int'>
```

- **`isinstance()` Function**: This is used to check if a variable matches a specific data type:

▼ Example Code

```
print(isinstance('Hello world', str)) # True
print(isinstance('John Doe', int)) # False
```

Working with Strings

- **Definition**: As you recall from JavaScript, strings are immutable which means you can not change them after they have been created. In Python, you can use either single or double quotes. It is recommended to chose a rule and stick with it:

▼ Example Code

```
developer = 'Jessica'
city = 'Los Angeles'
```

- **Accessing Characters from Strings**: You can access characters from strings by using bracket notation like this:

▼ Example Code

```
my_str = "Hello world"

print(my_str[0])  # H
print(my_str[6])  # w

print(my_str[-1])  # d
print(my_str[-2]) # l
```

- **Escaping Strings**: You can use a backslash (`\`) if your string contains quotes like this:

▼ Example Code

```
msg = 'It\'s a sunny day'
quote = "She said, \"Hello!\""
```

Saved! Your code was saved to your browser's local storage.

```
developer = 'Jessica'
print('My name is ' + developer + '.') # My name is Jessica
```

Another way to concatenate strings is by using the `+=` operator. This is used to perform concatenation and assignment in the same step like this:

▼ Example Code

```
greeting = 'My name is '
developer = 'Jessica.'

greeting += developer
print(greeting) # My name is Jessica.
```

- `f-strings`: This is short for formatted string literals. It allows you to handle interpolation and also do some concatenation with a compact and readable syntax:

▼ Example Code

```
developer = 'Jessica'
greeting = f'My name is {developer}.'
print(greeting) # My name is Jessica.
```

- **String Slicing**: This is when you can extract portions of a string. Here is the basic syntax:

▼ Example Code

```
str[start:stop:step]
```

The start position represents the index where the extraction should be begin. The stop position is where the slice should end. This position is non inclusive. The step position represents the interval to increment for the slicing. Here are some examples:

▼ Example Code

```
message = 'Python is fun!'

print(message[0:6])  # Python
print(message[7:])  # is fun!
print(message[::2])  # Pto sfn
```

- **Getting the Length of a String**: The `len()` function is used to return the number of the characters in the string:

▼ Example Code

```
developer = 'Jessica'

print(len(developer)) # 7
```

Working with the `in` operator

- `in` **Operator**: This returns a boolean that specifies whether the character or characters exist in the string or not:

▼ Example Code

```
my_str = 'Hello world'

print('Hello' in my_str)  # True
print('hey' in my_str)    # False
```

Saved! Your code was saved to your browser's local storage.

Common String Methods

- `str.upper()`: This returns a new string with all characters converted to uppercase:

▼ Example Code

```
developer = 'Jessica'

print(developer.upper()) # JESSICA
```

- `str.lower()`: This returns a new string with all characters converted to lowercase:

▼ Example Code

```
developer = 'Jessica'

print(developer.lower()) # jessica
```

- `str.strip()`: This returns a copy of the string with specified leading and trailing characters removed (if no argument is passed tc the method, it removes leading and trailing whitespace).

▼ Example Code

```
greeting = '  hello world  '

trimmed_my_str = greeting.strip()
print(trimmed_my_str)  # 'hello world'
```

- `replace()`: This returns a new string with all occurrences of the old string replaced by a new one.

▼ Example Code

```
greeting = 'hello world'

replaced_my_str = greeting.replace('hello', 'hi')
print(replaced_my_str)  # 'hi world'
```

- `split()`: This is used to split a string into a list using a specified separator. A separator is a string specifying where the split shoul happen.

▼ Example Code

```
dashed_name = 'example-dashed-name'

split_words = dashed_name.split('-')
print(split_words)  # ['example', 'dashed', 'name']
```

- `join()`: This is used to join elements of an iterable into a string with a separator. An iterable is a collection of elements that can b looped over like a list, string or a tuple.

▼ Example Code

```
example_list = ['example', 'dashed', 'name']

joined_str = ' '.join(example_list)
print(joined_str)  # example dashed name
```

- `str.startswith(prefix)`: This returns a boolean indicating if a string starts with the specified prefix:

```python
result = developer.startswith('N')
print(result)  # True
```

- `str.endswith(suffix)`: This returns a boolean indicating if a string ends with the specified suffix:

▼ Example Code

```python
developer = 'Naomi'

result = developer.endswith('N')
print(result)  # False
```

- `str.find()`: This returns the index for the first occurrence of a substring. If one is not found, then `-1` is returned:

▼ Example Code

```python
developer = 'Naomi'

result = developer.find('N')
print(result)  # 0

city = 'Los Angeles'
print(city.find('New')) # -1
```

- `str.count(substring)`: This counts how many times a substring appears in a string:

▼ Example Code

```python
city = 'Los Angeles'
print(city.count('e')) # 2
```

- `str.capitalize()`: This returns a new string with the first character capitalized and the other characters lowercased:

▼ Example Code

```python
dessert = 'chocolate cake'
print(dessert.capitalize()) # Chocolate cake
```

- `str.isupper()`: This returns `True` if all letters in the string are uppercase and `False` if otherwise:

▼ Example Code

```python
dessert = 'chocolate cake'
print(dessert.isupper()) # False
```

- `str.islower()`: This returns `True` if all letters in the string are lowercase and `False` if otherwise:

▼ Example Code

```python
dessert = 'chocolate cake'
print(dessert.islower()) # True
```

- `str.title()`: This returns a new string with the first letter of each word capitalized:

▼ Example Code

```python
city = 'los angeles'
print(city.title()) # Los Angeles
```

```
trans_table = str.maketrans('abc', '123')
print(trans_table) # {97: 49, 98: 50, 99: 51}

result = 'abcabc'.translate(trans_table)
print(result)  # 123123
```

Common Operations used with Integers and Floats

- **Basic Math Operations**: In Python, you can do basic math operations with integers and floats including addition, subtraction, multiplication and division:

▼ Example Code

```
int_1 = 56
int_2 = 12
float_1 = 5.4
float_2 = 12.0

# Addition

print('Integer Addition:', int_1 + int_2) # Integer Addition: 68
print('Float Addition:', float_1 + float_2) # Float Addition: 17.4

# Subtraction

print('Int Subtraction:', int_1 - int_2) # Int Subtraction: 44
print('Float Subtraction:',  float_2 - float_1) # Float Subtraction: 6.6

# Multiplication

print('Int Multiplication:', int_1 * int_2) # Int Multiplication: 672
print('Float Multiplication:', float_2 * float_1) # Float Multiplication: 64.80000000000001

# Division

print('Int Division:', int_1 / int_2) # Int Division: 4.666666666666667
print('Float Division:', float_2 / float_1) # Float Division: 2.2222222222222222
```

When you add a float and an integer, the result will be converted to a float like this:

▼ Example Code

```
int_1 = 56
float_1 = 5.4

print(int_1 + float_1) # 61.4
```

- **Modulus Operator (`%`)**: This returns the remainder when a number is divided by another number:

▼ Example Code

```
int_1 = 56
int_2 = 12

print(int_1 % int_2) # 8
```

```python
int_1 = 56
int_2 = 12

print(int_1 // int_2) # 4
```

- **Exponentiation Operator (`**`)**: This operator is used to raise a number to the power of another:

▼ **Example Code**

```python
int_1 = 4
int_2 = 2

print(int_1 ** int_2) # 16
```

- `float()` **Function**: You can use this function to convert an integer to float.

▼ **Example Code**

```python
num = 4

print(float(num)) # 4.0
```

- `int()` **Function**: You can use this function to convert an float to an integer.

▼ **Example Code**

```python
num = 4.0

print(int(num)) # 4
```

- `round()` **Function**: This is used to round a number to the nearest whole integer:

▼ **Example Code**

```python
num_1 = 3.4
num_2 = 7.7

print(round(num_1)) # 3
print(round(num_2)) # 8
```

- `abs()` **Function**: This is used to return the absolute value of a number:

▼ **Example Code**

```python
num = -13

print(abs(num)) # 13
```

- `bin()` **Function**: This is used to convert an integer to its binary representation as a string:

▼ **Example Code**

```python
num = 56

print(bin(num))  # 0b111000
```

- `oct()` **Function**: This is used to convert an integer to its octal representation as a string:

▼ **Example Code**

Saved! Your code was saved to your browser's local storage.

```
print(oct(num))  # 0o70
```

- `hex()` **Function**: This is used to convert an integer to its hexadecimal representation as a string:

▼ Example Code

```
num = 56

print(hex(num))  # 0x38
```

- `pow()` **Function**: This is used to raise a number to the power of another:

▼ Example Code

```
result = pow(2, 3)
print(result)  # 8
```

Augmented Assignments

- **Definition**: Augmented assignment combines a binary operation with an assignment in one step. It takes a variable, applies an operation to it with another value, and stores the result back into the same variable.

▼ Example Code

```
# Addition assignment
my_var = 10
my_var += 5

print(my_var) # 15

# Subtraction assignment
count = 14
count -= 3

print(count) # 11

# Multiplication assignment
product = 65
product *= 7

print(product) # 455

# Division assignment
price = 100
price /= 4

print(price) # 25.0

# Floor Division assignment
total_pages = 23
total_pages //= 5

print(total_pages) # 4

# Modulus assignment
bits = 35
bits %= 2
```

Saved! Your code was saved to your browser's local storage.

```
# Exponentiation assignment
power = 2
power **= 3

print(power) # 8
```

There are other augmented assignment operators too, like those for bitwise operators. They include `&=`, `^=`, `>>=`, and `<<=`.

Working with Functions

- **Definition**: Functions are reusable pieces of code that take inputs (arguments) and returns an output. To call a function, you need t reference the function name followed by a set of parenthesis:

▼ Example Code

```
# Defining a function

def get_sum(num_1, num_2):
    return num_1 + num_2

result = get_sum(3, 4) # function call
print(result) # 7
```

If a function does not explicitly return a value, then the default return value is `None`:

▼ Example Code

```
def greet():
    print('hello')

result = greet() # hello
print(result) # None
```

You can also supply default values to parameters like this:

▼ Example Code

```
def get_sum(num_1, num_2=2):
    return num_1 + num_2

result = get_sum(3)
print(result) # 5
```

If you call the function without the correct number of arguments, you will get a `TypeError`:

▼ Example Code

```
def calculate_sum(a, b):
    print(a + b)

calculate_sum()

# TypeError: calculate_sum() missing 2 required positional arguments: 'a' and 'b'
```

Common Built-in Functions

- `input()` **Function**: This is used to prompt the user for some input:

```
print('Hello', name) # Hello Kolade
```

- `int()` **Function**: This is used to convert a number, boolean, or a numeric string into an integer:

▼ Example Code

```
print(int(3.14)) # 3
print(int('42')) # 42
print(int(True)) # 1
print(int(False)) # 0
```

Scope in Python

- **Local Scope**: This is when a variable declared inside a function or class can only be accessed within that function or class.

▼ Example Code

```
def my_func():
    num = 10
    print(num)
```

- **Enclosing Scope**: This is when a function that's nested inside another function can access the variables of the function it's nested within.

▼ Example Code

```
def outer_func():
    msg = 'Hello there!'

    def inner_func():
        print(msg)
    inner_func()

print(outer_func()) # Hello there!
```

- **Global Scope**: This refers to variables that are declared outside any functions or classes which can be accessed from anywhere in the program.

▼ Example Code

```
tax = 0.70

def get_total(subtotal):
    total = subtotal + (subtotal * tax)
    return total

print(get_total(100))  # 170.0
```

- **Built-in Scope**: Reserved names in Python for predefined functions, modules, keywords, and objects.

▼ Example Code

```
print(str(45)) # '45'
print(type(3.14)) # <class 'float'>
print(isinstance(3, str)) # False
```

Comparison Operators

- **Equal** (`==`): Checks if two values are equal:

- **Not equal (`!=`)**: Checks if two values are not equal:

▼ Example Code

```
print(3 != 4) # True
```

- **Strictly greater than (`>`)**: Checks if one value is greater than another:

▼ Example Code

```
print(3 > 4) # False
```

- **Strictly less than (`<`)**: Checks if one value is less than another:

▼ Example Code

```
print(3 < 4) # True
```

- **Greater than or equal(`>=`)**: Checks if one value is greater than or equal to another:

▼ Example Code

```
print(3 >= 4) # False
```

- **Less than or equal(`<=`)**: Checks if one value is less than or equal to another:

▼ Example Code

```
print(3 <= 4) # True
```

Working with `if`, `elif` and `else` Statements

- **`if` Statements**: These are conditions used to determine if something is true or not. If the condition evaluates to `True`, then that block of code will run.

▼ Example Code

```
age = 18

if age >= 18:
    print('You are an adult') # You are an adult
```

- **`elif` Statement**: These are conditions that come after an `if` statement. If the `elif` condition evaluates to `True`, then that block of code will run.

▼ Example Code

```
age = 16

if age >= 18:
    print('You are an adult')
elif age >= 13:
    print('You are a teenager')  # You are a teenager
```

- **`else` Clause**: This will run if no other conditions evaluate to `True`.

▼ Example Code

Saved! Your code was saved to your browser's local storage.

```
if age >= 18:
    print('You are an adult')
elif age >= 13:
    print('You are a teenager')
else:
    print('You are a child')  # You are a child
```

You can also use nested `if` statements like this:

▼ Example Code

```
is_citizen = True
age = 25

if is_citizen:
    if age >= 18:
        print('You are eligible to vote') # You are eligible to vote
else:
    print('You are not eligible to vote')
```

Truthy and Falsy Values

- **Definition**: In Python, every value has an inherent boolean value, or a built-in sense of whether it should be treated as `True` or `False` in a logical context. Many values are considered truthy, that is, they evaluate to `True` in a logical context. Others are falsy, meaning they evaluate to `False`. Here are some examples of falsy values:

▼ Example Code

```
None
False
Integer 0
Float 0.0
Empty strings ''
```

Other values like non-zero numbers, and non-empty strings are truthy.

Working with the `bool()` Function

- **Definition**: If you want to check whether a value is truthy or falsy, you can use the built-in `bool()` function. It explicitly converts a value to its boolean equivalent and returns `True` for truthy values and `False` for falsy values. Here are a few examples:

▼ Example Code

```
print(bool(False)) # False
print(bool(0))  # False
print(bool('')) # False

print(bool(True)) # True
print(bool(1)) # True
print(bool('Hello')) # True
```

Boolean Operators and Short-circuiting

- **Definition**: These are special operators that allow you to combine multiple expressions to create more complex decision-making logic in your code. There are three Boolean operators in Python: `and`, `or`, and `not`.
- **`and` Operator**: This operator takes two operands and returns the first operand if it is falsy, otherwise, it returns the second operand. Both operands must be truthy for an expression to result in a truthy value.

▼ Example Code

```python
print(is_citizen and age) # 25
```

You can also use the `and` operator in conditionals like this:

▼ Example Code

```python
is_citizen = True
age = 25

if is_citizen and age >= 18:
    print('You are eligible to vote') # You are eligible to vote
else:
    print('You are not eligible to vote')
```

- `or` **Operator**: This operator returns the first operand if it is truthy, otherwise, it returns the second operand. An or expression results in a truthy value if at least one operand is truthy. Here is an example:

▼ Example Code

```python
age = 19
is_employed = False

print(age or is_employed) # 19
```

Just like with the `and` operator, you can use the `or` operator in conditionals like this:

▼ Example Code

```python
age = 19
is_student = True

if age < 18 or is_student:
    print('You are eligible for a student discount') # You are eligible for a student discount
else:
    print('You are not eligible for a student discount')
```

- **Short-circuiting**: The `and` and `or` operators are known as a short-circuit operators. Short-circuiting means Python checks values from left to right and stops as soon as it determines the final result.
- `not` **Operator**: This operator takes a single operand and inverts its boolean value. It converts truthy values to `False` and falsy values to `True`. Unlike the previous operators we looked at, `not` always returns `True` or `False`. Here are some examples:

▼ Example Code

```python
print(not '') # True, because empty string is falsy
print(not 'Hello') # False, because non-empty string is truthy
print(not 0) # True, because 0 is falsy
print(not 1) # False, because 1 is truthy
print(not False) # True, because False is falsy
print(not True) # False, because True is truthy
```

Here is an example of the `not` operator in a conditional:

▼ Example Code

```python
is_admin = False
```

Saved! Your code was saved to your browser's local storage.

```
print('Welcome, Administrator!')
```

Assignment

☐ Review the Python Basics topics and concepts.

Please complete the assignment

Submit

Ask for Help