Lecture 2
MTH 3300 Spr 2025

Jaime Abbariao

## Recap

### REPL

To open a Python REPL (read-eval-print loop), you have to go to your command line/terminal and type `python`

In this environment, you can evaluate different Python statements without having to use the `print` function.

## Recap

### Running a file

If you're using Visual Studio Code, there should be a play button somewhere on the top right that allows you to run your file.

In most cases, this is all you will need to run a file.

Otherwise, you'll need to run the following command:

```
python [insert name of your file here]
```

## Recap

### Types

- `int`: integers
- `float`: real numbers (like 3.14, 1.0, etc)
- `bool`: boolean values (`True` and `False`)
- `str`: strings or sequence of characters

## Recap

### Variables

Variables are ways to store information that we can reference or mutate in a program.

```
a_number = 40
a_string = "John"
a_bool = False
an_expression = 5 * 5
a, b = 10, 11
a, b, c = 10, 11, 12
```

Recap

Variables

Exercise

Create three variables and print each one out separately.

- `name` - `str`
- `age` - `int`
- `height` - `float`

Suppose that we have 4 variables `a`, `b`, `c`, and `d`. Swap the values of the following pairs using inline assignments:

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

## Recap

### Type casting

Type casting is a means of transforming data from one type to another

- `int()` can be used to transform some values into an integer
- `float()` can be used to transform some values into a real number
- `str()` can be used to transform some values into a string
- `bool()` can be used to transform some values into a boolean

## Recap

### Type casting

#### bool()

Certain values when cast into a boolean will evaluate to `False`. These values from the basic data types we know are:

- `0`
- `0.0`
- `""`

## Recap

## Type casting

## Exercise

- Create a variable `num_str` with the value "42".
- Convert it to an integer and store it in `num_int`.
- Convert it to a float and store it in `num_float`.
- Print all three values along with their types.

## Recap

### Input and output

To output items to the console when you run your code, you should use the `print()` function.

```
print("my name here")
```

## String interpolation

We can insert variables into strings by using Python's f-string feature.

Note here that we need to add `f` before the string to denote that the string is an f-string.

```
name = "Jaime"

print(f"Hello, {name}")
```

# Recap

## Input and output

In order to get user-inputted information, we can use the `input()` function.

The `input()` function always gives us a string.

If we need the input to be an integer or a real number, then we should cast them appropriately

```
age = input("How old are you? ")
age = int(age)
```

## Recap

## Input and output

## Exercise

Create a small program that receives a string `name` and two `float` numbers `monthly_salary` and `monthly_expenses` from the user.

Find out how much money this person has left and output the following: `<name> has $<remaining_amount> left for this month`.

## Arithmetic Operators

### Addition

When working with numbers (`int`, `float`), the addition operator `+` predictably will add the two numbers together.

```
_sum = 1 + 2
print(_sum)
# expected: 3
```

## Arithmetic Operators

### Addition

When working with strings (`str`), the addition operator will concatenate the strings together.

```
"Hello, " + "World!"
# expected: "Hello, World!"
```

## Arithmetic Operators

### Subtraction

The subtraction operator `-` will find the difference between two numbers.

```
_difference = 4 - 2
print(_difference)
```

Strings do not implement the subtraction operator. Using it with strings will raise an error.

## Arithmetic Operators

### Multiplication

The multiplication operator `*` will find the product of two numbers.

```
product = 12 * 12
print(product)
```

## Arithmetic Operators

### Multiplication

You cannot multiply two strings together; however you can multiple a string by an integer!

```
print("he" * 4)
# expected: hehehehe
```

This allows us to duplicate a sequence by the number of times we've denoted on the right-hand side.

## Arithmetic Operators

### Division

The division operator `/` will find the quotient between two numbers; however, the result will be of type `float`

```
print(4 / 2) # expected: 2.0
print(1 / 1) # expected: 1.0
print(1 / 0) # raises a ZeroDivisionError
```

## Arithmetic Operators

### Division

#### Integer Division

If we want to divide two integers and receive a quotient of type `int`, we use the integer division operator `//`

```
print(4 // 2) # expected: 2
print(7 // 2) # expected: 3
print(-1 // 2) # expected: -1
```

Integer division will always round down.

We should note that integer division only respects the integer result if and only if both parts of the expression are integers.

For example, `print(5.8 // 2)` will actually print out `2.0` rather than `2` because `5.8` has type `float`

## Arithmetic Operators

### Modulo

The modulo operator `%` returns the remainder after a number if divided by another.

In Python, the modulo operator follows the following implementation:

```
a % b = a - (b * floor(a / b))
```

```
print(7 % 3)   # expected: 1
print(-7 % 3)  # expected: 2
print(7 % -3)  # expected: -2
print(-7 % -3) # expected: -1
```

Note: `floor` function will round a value down.

## Arithmetic Operators

### Exponentiation

Exponentiation is done using the `**` operator

```
print(2 ** 128)
print((-3) ** 3)
```

## 🟦 Python Lab Exercise: Arithmetic, Logical, and Relational Operators

### 🟩 Task 1: Arithmetic Operators

Write a Python program that performs the following calculations:

1. Ask the user to input two numbers, `num1` and `num2`.
2. Perform the following operations and display the results:
   - Addition
   - Subtraction
   - Multiplication
   - Division
   - Floor division
   - Modulo
   - Exponentiation

Example Output:

```
Enter the first number: 10
Enter the second number: 3
Addition: 13
Subtraction: 7
Multiplication: 30
Division: 3.333
Modulo: 1
Exponentiation: 1000
Floor Division: 3
```

## Relational Operators

### Equals

The equals operator `==` checks if two values are equal.

If they are equal, then it returns `True` otherwise it returns `False`.

```
print(100 == 2)
print("name" == "name")
print(True == 1)
print(False == 0)
```

## Relational Operators

### Equals

#### Issue with comparing floating point numbers

```
print(0.3 == 0.3) # expected: true

# But how about
print(0.1 + 0.2 == 0.3)
```

In Python, `0.1 + 0.2 == 0.3` actually returns false because `0.1 + 0.2` is `0.30000000000000004`.

This happens for the following reasons:

- 0.1 and 0.2 have infinitely repeating binary representations
- Since Python typically uses 64 bits to store a `float` value, this means that the infinitely repeating binary representation will have to be truncated
- The sum of two truncated values then may not necessarily lead to a precise value!

## Relational Operators

### Equals

#### Issue with comparing floating point numbers

A best practice for comparing `float`s is then check that the absolute difference between the values is less than some tolerance.

```
epsilon = 0.000000001

a = 0.1 + 0.2
b = 0.3

print(abs(a - b) < epsilon)
```

Note here that `abs()` gives us the absolute value of a number

## Relational Operators

### Not equals

The `!=` operator checks whether two values are not equal to each other.

```
print(0.3 != 0.3)
print("cheese" != "egg")
print(1 != 1.0)
```

## Relational Operators

### Less than | Less than or equal

The `<` operator returns `True` if the left-hand side is less than the right-hand side and `False` otherwise.

The `<=` operator returns `True` if the left-hand side is less than or equal to the right-hand side and `False` otherwise.

```
print(1 < 12)
print(123 <= 123)
```

## Relational Operators

### Greater than | Greater than or equal

The `>` operator returns `True` if the left-hand side is greater than the right-hand side and `False` otherwise.

The `>=` operator returns `True` if the left-hand side is greater than or equal to the right-hand side and `False` otherwise.

```
print(1 > 12)
print(123 >= 123)
```

## Python Lab Exercise: Arithmetic, Logical, and Relational Operators

### Task 2: Relational Operators

Write a Python program that compares two numbers, `num1` and `num2`, using relational operators.

1. Ask the user to input two numbers.
2. Compare the numbers and display the results of the following comparisons:
   - Greater than (`num1 > num2`)
   - Less than (`num1 < num2`)
   - Equal to (`num1 == num2`)
   - Not equal to (`num1 != num2`)
   - Greater than or equal to (`num1 >= num2`)
   - Less than or equal to (`num1 <= num2`)

Example Output:

```
Enter the first number: 5
Enter the second number: 8
5 > 8: False
5 < 8: True
5 == 8: False
5 != 8: True
5 >= 8: False
5 <= 8: True
```

## Logical Operators

### and

| Operand A | Operand B | A AND B |
|---|---|---|
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

```
print(True and True)   # Output: True
print(True and False)  # Output: False
print(False and True)  # Output: False
print(False and False) # Output: False
```

## Logical Operators

### or

| Operand A | Operand B | A OR B |
|---|---|---|
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

```
print(True or True)   # Output: True
print(True or False)  # Output: True
print(False or True)  # Output: True
print(False or False) # Output: False
```

## Logical Operators

### not

The `not` operator negates the boolean value of a variable/expression/constant

```
some_bool = True

print(not some_bool) # expected: False
```