# ***WEEK 1***

## Strings

- Note:
  - A "safer" way again (in terms of inadvertently ending the string mid-string) to format strings is by using a triple quotation mark at the start and end of the string, e.g. """Hello, world""" or '''Hello, world'''. We will see lots of triple-quoted strings later in the subject, in the context of documenting functions.
  - Can also be used to comment over multiple lines instead of '#'
- **Concatenation:**

```
print("Hello" + "World")

HelloWorld
```

```
print('Hi' + 'Hi' + 'Hi')
print('Hi' * 3)

HiHiHi
HiHiHi
```

## Variables

- Note 1:
  - When a variable is assigned a new value, its old value is forgotten
- Note 2:
  - The following words having special meaning in Python, and so they cannot be used for variable names: and, as, assert, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, None , nonlocal, not, or, pass, raise, return, True, try, while, with, yield.

## Long Lines

- If you want a statement to continue over multiple lines you just write a backslash \ at the end of the line, and continue writing.

## The Input Function

- Python has a built-in function called `input` that can be used to get keyboard input from the user as a string.
- The `input` function can be given a message to display, usually prompting the user with what kind of information the program wants

```
name = input("What is your name? ")
age = input("What is your age? ")
print("I know that", name, "is", age, "years old!")
```

```
What is your name? Chris
What is your age? 21
I know that Chris is 21 years old!
```

- We have called the function with multiple arguments (multiple strings).
- `print` combines the arguments together into a single output, separating each adjacent pair or arguments with a space.

## Expressions

- Basic Format:
  - Operand 1 operator operand2
- the assignment statement, e.g. x = 5, is not an expression and therefore can't be used at a position in your program where you need a value:
  - y = 3 + (x = 3) = NOT ALLOWED

## Types of operands

| Type | Description |
|---|---|
| `int` | For whole numbers eg: `-3`, `-5`, or `10` |
| `float` | For real numbers eg: `-3.0`, `0.5`, or `3.14159` |
| `bool` | The Boolean type. For storing `True` and `False` (only those two values; Booleans allow for no grey areas!). |
| `str` (= "string") | For chunks of text, e.g.: `"Hello, I study Python"` |
| `tuple` | For combinations of objects, e.g.: `(1, 2, 3)` or `(1.0, "hello", "frank")` |
| `list` | A more powerful way of storing lists of objects, e.g. `[1, 3, 4]` or `[1.0, "hello", "frank"]` |
| `dict` | We will see this later ... maybe you can guess what it does, e.g. `{"bob": 34, "frankenstein": 203}` |

### Expressions with Integers and Floats

| Expression | Result |
|---|---|
| `int`(*,+,-)`int` | `int` |
| `int`/`int` | `float` |
| `float`(*,+,-,/)`float` | `float` |
| `float`(*,+,-,/)`int` | `float` |
| `int`(*,+,-,/)`float` | `float` |

- To check variable type:
- Print ( type(variable) )

## Floating point numbers are approximations

- Contain rounding errors, and thus are only approximations
- A rounding error is the discrepancy between the approximated value and its exact value.
- the `float` data type has only a fixed number of bits available, some numbers will inevitably have rounding errors.

## Type Conversion

- Python provides an explicit means of converting one data type into another data type (if the conversion is possible).
- This conversion is also called type casting

```
print(int(32.7))
32

print(int("32"))
32
```

- Unlike `int` casting, a cast to `float` can convert any string representation of a floating-point number, even if the number contains a decimal point or is written in scientific notation.

```
print(float(32))
32.0
```

- The function `str` converts values to the type `str` (a string).

```
print(str(32) + str(32.7))
3232.7
```

## Input for numeric types

- `input` always returns a string, so if your program requires a string as input, there is nothing more to do.
- As `input` always returns strings, even if you type a number, Python will see this as a string consisting of digits

```
num = input('Enter a number to double: ')
print('2 x', num, '=', 2 * num)

Enter a number to double: 6
2 x 6 = 66
```

- "num" contains 33 characters as part of a string, therefore 2* 33 returned

- We need a function that converts a string where each character is a digit, into a number. If your program expects integer values, you simply use the function `int`:

```
num = int(input('Enter a number to double: '))
print('2 x', num, '=', 2 * num)

Enter a number to double: 6
2 x 6 = 12
```

## Integer Division

- **Use** `//`
- in integer division, the result is always rounded down to the nearest integer

## Modulo

- Use `%`
- The modulus operator computes the remainder of an integer division

## Exponentiation

- Use `**`
- to raise things to powers in Python we use the exponentiation operator