TECHNION
Azrieli Continuing Education and External Studies Division

# Module 5.1.2: Variables, Expressions and Syntax

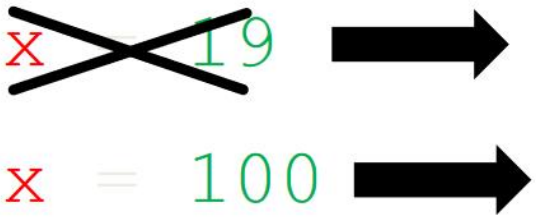
# Variables

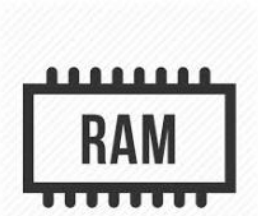
- Variables are used to store data.
- The data of the variable is stored in the memory.
- The data that is stored in the variable can be changed during runtime.
- In order to store data inside a variable, we need two parameters.
  - The Name of the variable.
  - The data that will be stored.

```
x = 12
y = 'Hello'
```

# Variables

- Variables are names used to refer to some memory locations of the computer.
- The data stored inside is referred to as the value of a variable.
- The variables are not fixed values and might change.
- Can not be used if not initialized.



# Python Variable Name Rules

- Must start with a letter or underscore.
- Must consist of letters and numbers and underscores.
- Cannot start with a number.
- Case Sensitive.

# Assignment Statements

- We assign a value to a variable using the assignment statement (=).
- An assignment statement consists of expression on the right-hand side and a variable to store the result on the left-hand.

# Numeric Expressions

# Numeric Expressions

- Because of the lack of mathematical symbols on computer keyboards - we use “computer-speak” to express classic math operations.
- Asterisk is used for multiplication → *
- Exponentiation (raise to a power) looks different from in math → **.

| Operator | Operation |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| ** | Power |
| % | Remainder |

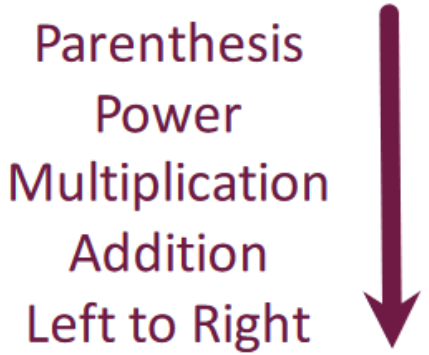
# Order of Evaluation

- When we string operators together - Python must know which one to do first.
- This is called “operator precedence”.
- Which operator “takes precedence” over the others?

```
x = 1 + 2 ** 3 / 4 * 5
```

# Operator Precedence Rules

- Highest precedence rule to lowest precedence rule:
  - Parenthesis → ( )
  - Exponentiation (raise to a power) → **
  - Multiplication, Division & Remainder
  - Addition & Subtraction
  - Left to Right



# Variable Modification

- x = 2 → A new variable (x) has been created with the value of 2.
- x = x + 2 → A new assignment for x has been written. In order to store the data inside the variable (x), the mathematical expression must be evaluated.
- Take the variable (x) before + current value (2) and add what is on the right of the equals sign (2) to the current value of what is before the + sign.

# Data Types

# Data Types In Python

| Numbers | | Strings | | |
|---|---|---|---|---|
| **Integers** | **Float** | **Sings Quotes** | **Double Quotes** | **Triple Quotes** |
| Whole Numbers | Floating point numbers | If character holds in a single, double or triple quotes, then we can call it a string. | | |
| Can be either negative or positive | Can be either negative or positive | Value holds in a Single-Quotes equals to a value holds in a Double-Quotes. You CANNOT mix between them. | | Can store Single, Double-quotes, Multi-line string. |
| Int | Float | Str | Str | Str |
| 1, -3, 450, 22 | 0.33, 1.0, 6.7754 | ‘Text’ | “Text” | “’Text’”’ |

# Using Operators on Different Data Types

- Using the + Operator on Numbers will result in **addition**.
- Using the + Operator on Strings will result in the **concatenation**.
- Some operations are prohibited. **We cannot add an integer to a string.**

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

# Casting

- Casting → Create an object from another type!
- By using a cast function, we can convert between strings and integers!
  - Cast functions for instance.
    - int()
    - float()
    - strings()

# The print() Function

# The print() Function

- The print function prints out any value to the screen.
- How could we have known that?
- The function help() to describes function's use.

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

# More Print Function Inputs

- The print function can receive more inputs. For example, we can configure the values that will separate different inputs. Or we can configure what we wish to have printed at the end of each print. The default is a newline character, but this can be changed to an empty string for the print to stay on the same line.

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

# User Input

# User Input

- We can instruct Python to pause and read data from the user using the input() function.

- The input() function returns a string.

```
name = input('Who are you? ')
print('Welcome', name)
```

# Converting User Input

- If we want to read a number from the user, then we must convert it from a string to a number using a type conversion function.

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

print('You are', age, 'years old.')
```

# Comments

# Comment in Python

- Comments are line that are ignored by the interpreter.

- There are two ways to create comments.
  - Every line that starts with # will be ignored by the interpreter.
  - Every lines between triple quotes are also ignored by the interpreter.

```
an.import_vm(an.config['VM_INFO']['ova_path'], an.config['VM_INFO']['vm_name'], an.config['VM_INFO']['vm_group'])
# print('Creating snapshot')
an.create_snapshot(an.config['VM_INFO']['vm_name'])

an.list_snapshots(an.config['VM_INFO']['vm_name'])

# print('Deleting snapshot')
# an.restore_snapshot(an.config['VM_INFO']['vm_name'], 'Clean System')
```

# Thank You!