15-110 Principles of Computing – S21

Lecture 6:

Data Types, Arithmetic operators

Teacher:
Gianni A. Di Caro

The structure of a python program: definitions and commands

- A python program (also termed a *script*) is a sequence of definitions and commands
  - Definitions are ***evaluated*** for correctness, but are not executed
  - Commands (also termed statements) are ***executed,*** one at-a-time

✓ Definition of a function object

```
def sum_of_two_numbers(x,y):
    s = x + y
    print('The sum is:', s)
    return s
```

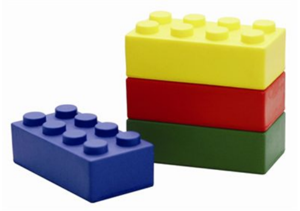
✓ Commands, that will be executed step-by-step

```
sum_is = sum_of_two_numbers(3, 5)
```

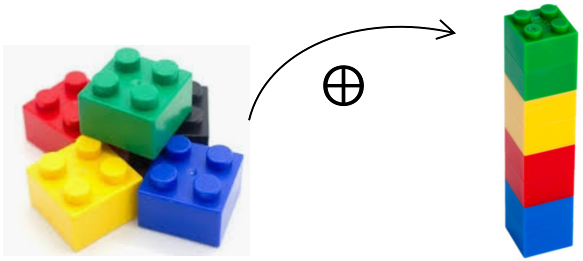
✓ Command, that executes the function defined before

The structure of a python program: objects, operators, expressions

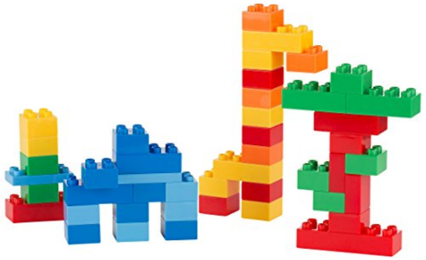
- Commands *instruct* (the python interpreter) to do something
  - ✓ A command includes **Objects** (words and concepts of our language)
  - ✓ A command can also include **Operators** to act upon the objects
  - ✓ Operators can be used to create **Expressions** that can be evaluated (i.e., generates a result)



Objects



Operators



Expressions

The structure of a python program: objects, operators, expressions

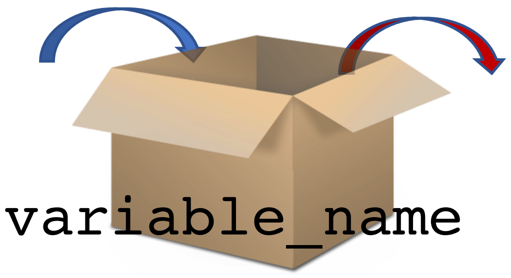
- Commands *instruct* (the python interpreter) to do something
  - ✓ A command includes **Objects** (words and concepts of our language)
  - ✓ A command can also include **Operators** to act upon the objects
  - ✓ Operators can be used to create **Expressions** that can be evaluated (i.e., generates a result)

```
3.5 + 2           Expression
x = 3             Assignment operator
y = 2 * 2.1       Expression + Assignment operator
print(x + y + 1)
```

Kind of objects

<u>Objects can be of two kinds:</u>

- ***Literal* objects** ↔ Only have a (fixed) ***value*** (e.g., 1, 3.5)
- ***Variable* objects** ↔ Have a value and a ***name***, an ***identifier*** (e.g., x = 1, y = 3.5, table = 'red')



Objects types

- Objects have a **type,** that defines the things that can be done (or not) with the object
  - ✓ With a car you can travel but not fly, with a cake you can eat but not travel ….)

- Numeric: to represent numbers of various type (e.g., `1, 3.5`)
- Character string: to represent textual information (e.g., “`Temperature is 35 degrees`”)
- Logical (binary) to represent truth of falsity of conditions (i.e., `True, False`)
- *Structured ways* to frame and represent groups of numbers, strings, and logical data …

Terminology: Scalar vs. Non-Scalar objects

An object type can be **composite**, made of multiple components, or be **indivisible**

- Scalar type (e.g., `1, 3.5`)
- Non-scalar type (e.g., `"Hello"`)

Vs.

✓ **Scalar**: the object is indivisible

✓ **Non-scalar**: the object is composed by multiple parts that can be individually manipulated (accessed, modified, removed, added)

Numeric (scalar) types

- Scalar type objects:
  - `int`: **Integer relative numbers ($\mathbb{Z}$)**
    - Examples of literals of type `int` are: 2, 3, -1, 1000, 2001, -99
  - `float`: **Real numbers ($\mathbb{R}$)**
    - Examples of literals of type `float` are: 2.0, 3.2, -1.5, 1000.0, 2001.002, -99.1, 1.6E3
    - Why are do they called *float* instead of *real*?

- How do you know what's the type of a variable object `x`?
  - Use the function `type(x)`

Operators for numeric types: +, -, *, /, **

Let `i` and `j` be two literals that can be either `int` or `float`

- **Sum:** `i+j`
  - Type of expression: `int` if both integers, `float` otherwise
- **Difference:** `i-j`
  - Type of expression: `int` if both integers, `float` otherwise
- **Product:** `i*j`
  - Type of expression: `int` if both integers, `float` otherwise
- **Division:** `i/j`
  - Returns the real-valued result of the division, type of expression: `float`
- **Power raising:** `i**j`
  - Returns the i^j , type of expression: `int` if both integers, `float` otherwise

Operators for numeric types: different types of division!

- Division between two numbers i, j: $\quad i \div j = n + \frac{r}{j} \;\rightarrow\; i = j \times n + r$

  where n is the <u>integer quotient</u> and r is the <u>remainder of the integer division</u>

  $n =$ how many times j precisely fits in i,

  $\frac{r}{j} =$ the fractional remainder after the integer division

4 / 3 = 1.3333 = 1 + 0.333

12 / 3 = 4 = 4 + 0

5 / 3 = 1.6666 = 1 + 0.666

Operators for numeric types: /, //, %

- **(real) Division:** `i/j` $= n + \frac{r}{j}$
  - 4/3 → 1.3333 1 / 2 → 0.5 4 / 2 → 2.0 (float!). 12.5 / 4 → 3.125
- **Integer division:** `i//j` returns the integer quotient, n, and ignores the fractional remainder
  - Floor division: it returns the nearest integer of the real division result
  - Type of expression: `int` if both integers, `float` otherwise
  - 12 // 3 → 4 13 // 3 → 4 4 // 3 → 1 2 // 3 → 0 4.5 // 3.2 →1.0 4//2 → 2 12.5 // 4 → 4.0
- **Modulus:** `i%j` returns the remainder r from the *integer division*
  - Type of expression: `int` if both integers, `float` otherwise
  - 12 % 3 → 0 13 % 3 → 1 4 % 3 → 1 2 % 3 → 2 4.5 // 3.2 →1.29 4 % 2 → 0 12.5 // 4 → 0.5

Useful functions for arithmetic

- **Absolute value** (discard the sign): `abs(x)`

  `abs(-2)` → `2`, `abs(4.5)` → `4.5`

- **Ceiling function** (round up to the nearest integer): `math.ceil(x)`

- **Flooring function** (round down to the nearest integer): `math.floor(x)`

➢ To use these functions, we need to *import* the module providing them

```
import math

x = 7.4

print(math.floor(x), math.ceil(x))  → 7 8
```

math.floor(1.5) → 1	math.ceil(1.5) → 2
math.floor(-1.5) → -2	math.ceil(-1.5) → -1
math.floor(1.999) → 1	math.ceil(1.999) → 2

Area of a circle rounded down to the nearest integer value?

Write the function `circle_area_round(r)` that takes as input the radius of the circle and returns the value of the area rounded down to the nearest integer value.

What’s the hour?

It’s 8:00 (am/pm doesn’t matter). Where will the hour hand be in 7 hours? In practice the hour will be 15, but we are restricted to show 3 since the hour hand only goes up to 12!

Write the function `my_watch(now, next)` that takes as input the current hour, `now` (as an integer) and `next` hours in the future and returns the hour in the future ha shown by the hour hand of a clock.