CSCA08 — Introduction to Computer Science

Python expressions — weeks 1 and 2

- arithmetic expressions:
  - numbers: evaluate to `int` or `float`
  - larger arithmetic expressions are built from smaller arithmetic expressions:
    expression1 operator expression2

    1. evaluate ("find") the operator
    2. evaluate the operands `expression1` and `expression2`
    3. apply the operator to the resulting values from step 2
    4. the value of the expression `expression1 operator expression2` is the value returned by the application of the operator in step 3

Python expressions — weeks 1 and 2

- function calls:
  func-name(arg1, arg2, ...) where each *arg* is an expression
  1. evaluate ("find") the function named `func-name`
  2. evaluate each argument, in order from left to right
  3. call the function `func-name` with the values obtained in step 2 as arguments
  4. the value of the expression `func-name(arg1, arg2, ...)` is the value returned by the function call in step 3

Python expressions — strs and operations on strs

- string literals (e.g., `'CSCA08'`) evaluate to objects of type `str`
- string concatenation using +:
  expression1 + *expression2*
  1. evaluate `expression1` and `expression2`: get two objects of type `str`
  2. create a new `str` that results from concatenating the two `strs` obtained in step 1
  3. the value of `expression1 + expression2` is the value resulting from the concatenation in step 2

Python expressions — strs and operations on strs

- string indexing:
  str-expression[index-expression]
  1. evaluate `str-expression`: get object of type `str`
  2. evaluate `index-expression`: get object of type `int`
  3. create a new `str` object: the character that appears in the `str` from step 1 at the index from step 2
  4. the value of `str-expression[index-expression]` is the `str` created in step 3

Python expressions — strs and operations on strs

- string slicing:
  str-expression`[`*from-index-expression*`:` *to-index-expression*`]`
  1. evaluate `str-expression`: get object of type `str`
  2. evaluate `from-index-expression`: get object of type `int`
  3. evaluate `to-index-expression`: get object of type `int`
  4. create a new `str` object that contains every character that appears in the string from step 1 starting at, and including, index from step 2 up to, and excluding, index from step 3
  5. the value of `str-expression[from-index-expression: to-index-expression]` is the `str` obtained in step 4

Python expressions — bools and operations on bools

- two boolean values: `True`, `False`
- operator `and`:
  expression1 and expression2
  1. evaluate `expression1`
  2. if the obtained value is `False`, the value of entire expression is `False`
  3. otherwise, evaluate `expression2`; the value of the entire expression is the value of `expression2`
- operator `or`:
  expression1 or expression2
  1. evaluate `expression1`
  2. if the obtained value is `True`, the value of entire expression is `True`
  3. otherwise, evaluate `expression2`; the value of the entire expression is the value of `expression2`