Ceng 111 – Fall 2021
Week 9a

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.

Previously on CENG111!

Tuples in Python

```
>>> (1, 2, 3, 4, "a")
(1, 2, 3, 4, 'a')
>>> type((1, 2, 3, 4, "a"))
<type 'tuple'>
```

- Tuples in Python: collection of data enclosed in parentheses, separated by comma.
- Accessing elements of a tuple (like strings):
  - Positive Indexing: `(1, 2, 3, 4, "a")[2]` returns 3.
  - Negative Indexing: `(1, 2, 3, 4, "a")[-1]` returns 'a'.
  - Ranged Indexing, *i.e.*, `[start:end:step]`: `(1, 2, 3, 4, "a")[0:4:2]` leads to `(1, 3)`.

Previously on CENG111!

Lists in Python

```
>>> [1, 2, 3, 4, "a"]
[1, 2, 3, 4, 'a']
>>> type([1, 2, 3, 4, "a"])
<type 'list'>
```

- Lists in Python: collection of data enclosed in brackets, separated by comma.
- Accessing elements of a list (like strings & tuples):

- Positive Indexing: `[1, 2, 3, 4, "a"][2]` returns 3.
- Negative Indexing: `[1, 2, 3, 4, "a"][-1]` returns `'a'`.
- Ranged Indexing, *i.e.*, `[start:end:step]`: `[1, 2, 3, 4, "a"][0:4:2]` leads to `[1, 3]`.

Previously on CENG111!

A frequent operation with containers

- Membership
  - in
  - not in
- “en” in “deneme”
  - True
- “an” in “deneme”
  - False
- “dem” in “deneme”
  - False

Previously on CENG111!

Variable Naming in Python

- Variable names are case sensitive. So, the names `a` and `A` are two different variables.
- Variable names can contain letters from the English alphabet, numbers and an underscore `_`.
- Variable names can only start with a letter or an underscore. So, `10a`, `$a`, and `var$` are all invalid whereas `_a` and `a_20`, for example, are valid names in Python.

- Variable names cannot be one of the keywords in Python:

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

Previously on CENG 111!

Variables, Values and Aliasing in Python

- Every data (whether constant or not) has an identifier (an integer) in Python:

```
>>> a = 1
>>> b = 1
>>> id(1)
135720760
>>> id(a)
135720760
>>> id(b)
135720760
```

- If the type of the data is mutable, there is a problem!!!

```
>>> a = ['a', 'b']
>>> b = a
>>> id(a)
3083374316L
>>> id(b)
3083374316L
>>> b[0] = 0
>>> a
[0, 'b']
```

This is called Aliasing.

Previously on CENG111!

How to make copy of the list?

- list(List-to-be-copied)
- L[:]
- Shallow copy

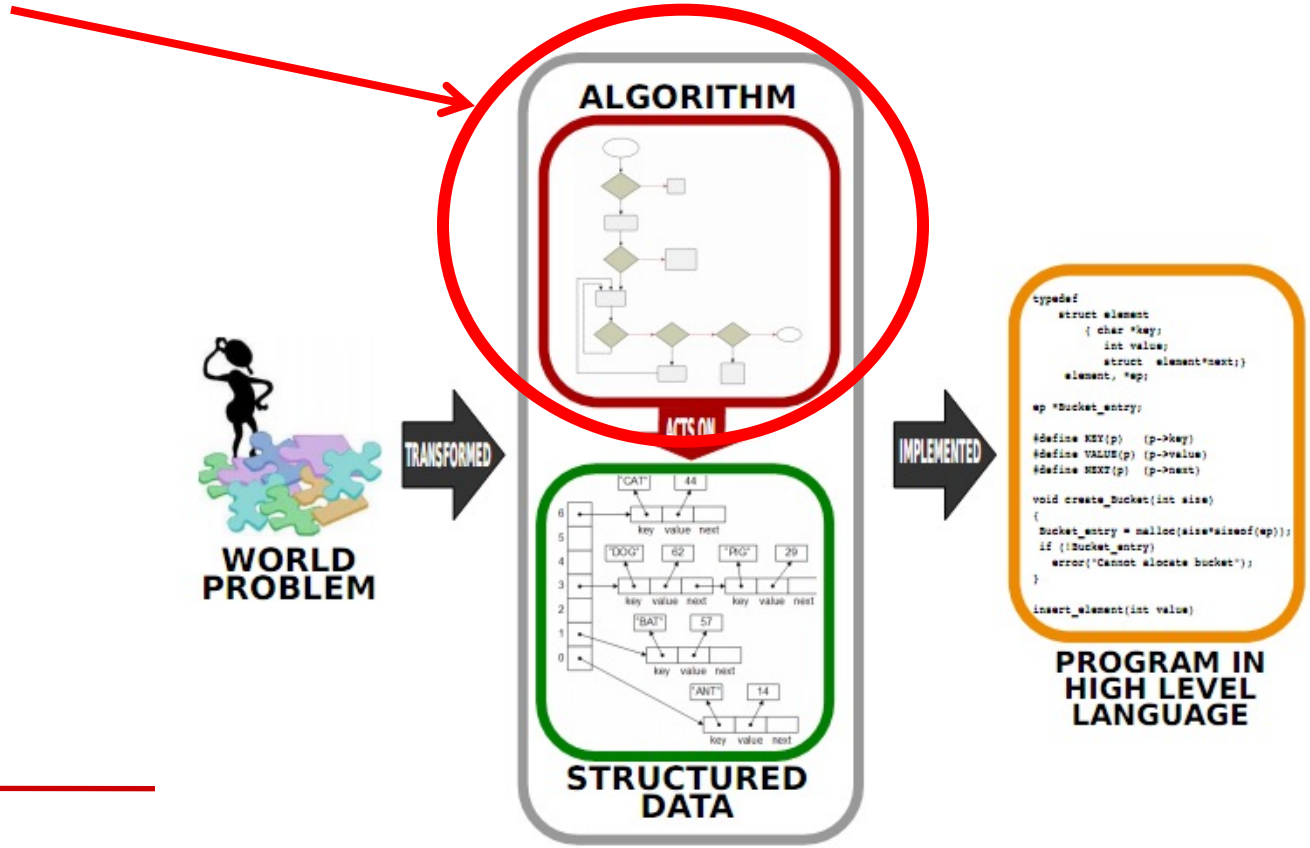
  import copy

  copy.copy(list)
- Deep copy

  import copy

  copy.deepcopy(list)

Previously on CENG111!



What are actions?

- Actions in a PL are the *things* that we can do with the data. What could they be?
  - Create data or modify data
  - Interact with the external environment

Previously on CENG111!

Action Types in High-Level Languages

- Expression evaluation
  - 3 + 4 * 5 / 2

vs.

- Statement execution
  - del L[2:4]

Today

- Expressions and their evaluation

Administrative Notes

- THE2 announced:
  - Due date: 26 December, 23:59
- Midterm:
  - 22 December, Wednesday, 18:00

Expressions

- An expression is a calculation which has a set of *operations*.
- Operations have *operators* and *operands*.
- Example: 3 + 4
  - + → operator
  - 3, 4 → operands

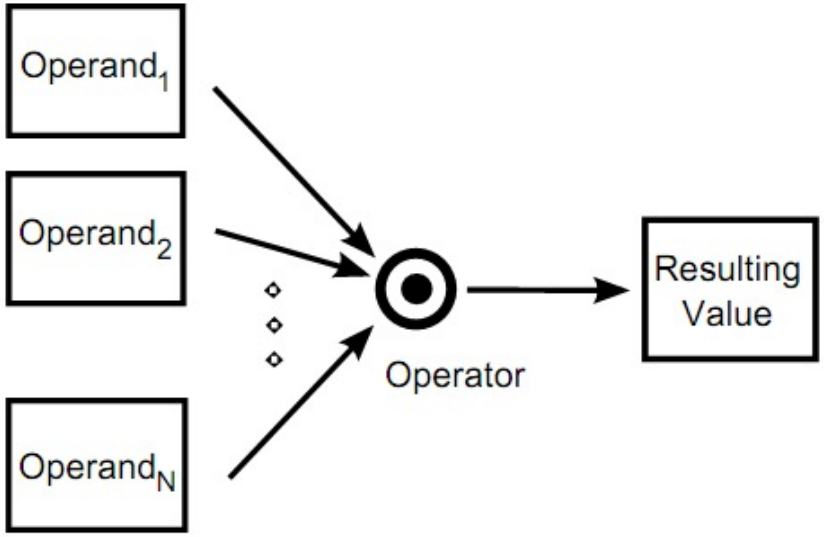
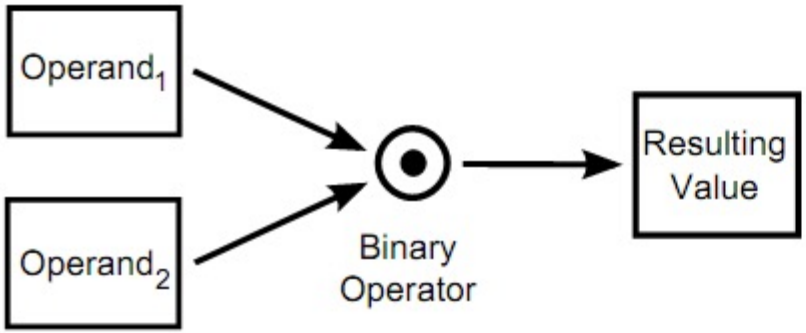
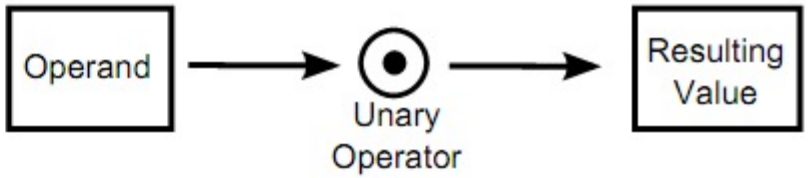


Figure 3.1: N-ary operation



(a) Binary operation



(b) Unary operation

Expressions in Python
- Involving arithmetic operators -

Operator	Operator Type	Description
+	Binary	Addition of two operands
-	Binary	Subtraction of two operands
-	Unary	Negated value of the operand
+	Unary	Positive value of the operand
*	Binary	Multiplication of two operands
/	Binary	Division of two operands
**	Binary	Exponentiation of two operands (Ex: `x**y` $= x^y$)

Expressions in Python
- Involving arithmetic operators -

- Precedence & Associativity of arithmetic operators.
- What is precedence?
  - The expression “3 + 4 * 5” has two different interpretations:
    - (3+4)*5
    - 3 + (4*5)
- What is associativity?
  - The expression “3.02 + 4.1 + 5.24” has two different interpretations:
    - (3.02+4.1)+5.24
    - 3.02+(4.1+5.24)

Expressions in Python
- Involving arithmetic operators -

- Precedence & Associativity of arithmetic operators.
- Top: highest precedence.
- Bottom: lowest precedence.

Operator	Type	Associativity	Description
**	Binary	Right-to-left	Exponentiation
+, -	Unary	Right-to-left	Positive, negative
*, /, //, %	Binary	Left-to-right	Multiplication, Division, Remainder, Modulo
+, -	Binary	Left-to-right	Addition, Subtraction

Floor division
(fraction part of the result is removed)

Expressions in Python
- Involving container operators -

- Concatenation (+)
  - “a” + “b” → “ab”
- Repetition (*)
  - “a” * 3 → “aaa”
- Membership (in, not in):
  - “a” in “Mathematics” → True
  - “a” not in “Mathematics” → False
- Indexing ([])

Expressions in Python
- Involving container operators -

- Precedence and associativity of container operators

Operator	Type	Associativity	Description
[]	Binary	Left-to-right	**Indexing**
**	Binary	Right-to-left	Exponentiation
+, -	Unary	Right-to-left	Positive, negative
*, /, //, %	Binary	Left-to-right	Multiplication & **Repetition**, Division, Remainder, Modulo
+, -	Binary	Left-to-right	Addition, Subtraction, **Concatenation**
in, not in	Binary	Right-to-left	Membership

Expressions in Python
- Involving relational operators -

- Equality (==)
  - Two data are equivalent if they represent the same value/information!
  - "Ali" == "Ali" → True
- Less-than (<):
  - A numerical data is less than another if the value of the first is less than that of the second:
    - 3 < 4.5 → True
  - A string is less than another if it is lexicographically (i.e., in ASCII value) less than the second.
    - "abc" < "def" → True
  - A tuple/list is less than another tuple/list if the first different items satisfy the less-than relation.

Expressions in Python
- Involving relational operators -

- Less-than-or-equal (<=)
  - <= → (<) or (==)
- Greater-than (>)
  - > → not (<=)
- Greater-than-or-equal-to (>=)
  - >= → not (<)
- Not-equal (!=)
  - != → not (==)

Note that in Python, relational operators can be chained. In other words, `a RO b RO c` (where `RO` is a relational operator) is interpreted as:

`(a RO b) and (b RO c)`.

In most other programming languages, `a RO b RO c` is interpreted as `(a RO b) RO c`.

Expressions in Python
- Involving relational operators -

- Precedence & Associativity

Operator	Type	Associativity	Description
[]	Binary	Left-to-right	**Indexing**
**	Binary	Right-to-left	Exponentiation
+, -	Unary	Right-to-left	Positive, negative
*, /, //, %	Binary	Left-to-right	Multiplication & Repetition, Division, Remainder, Modulo
+, -	Binary	Left-to-right	Addition, Subtraction, Concatenation
in, not in, <, <= >, >=, ==, !=	Binary	Right-to-left	Membership, **Comparison**

Expressions in Python
- Involving logical operators -

- Logical operators manipulate truth values:
- and operator
  - A and B → True iff (A is True) & (B is True)
- or operator
  - A or B → True iff either (A is True) or (B is True)
- not operator
  - not A → True iff A is False

Expressions in Python
- Involving logical operators -

- Precedence & Associativity

Operator	Type	Associativity	Description
[]	Binary	Left-to-right	**Indexing**
**	Binary	Right-to-left	Exponentiation
+, -	Unary	Right-to-left	Positive, negative
*, /, //, %	Binary	Left-to-right	Multiplication & Repetition, Division, Remainder, Modulo
+, -	Binary	Left-to-right	Addition, Subtraction, Concatenation
in, not in, <, <= >, >=, ==, !=	Binary	Right-to-left	Membership, Comparison
not	Unary	Right-to-left	Logical negation
and	Binary	Left-to-right	Logical AND
or	Binary	Left-to-right	Logical OR

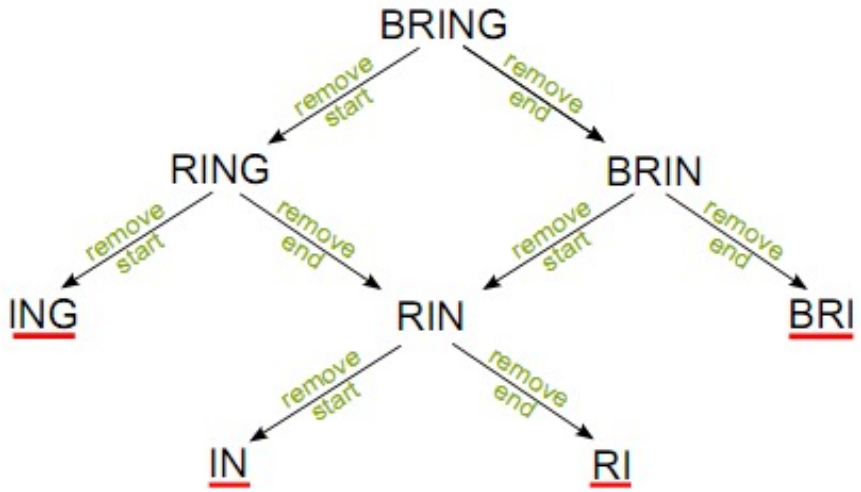
Expressions in Python
- assignment (**not** an operator) -

- Single assignment:
  - a = 4
- Multiple assignment:
  - a = b = c = 4
- Combined assignment:
  - a = a + 4 → a += 4
  - +=, *=, -=, /=, etc.

```
>>> b += 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
>>> b = 5
>>> b **= 2
>>> b
25
```

Church-Rosser Property

- A reduction/re-writing system has the Church-Rosser Property if the set of rules always lead to the same results independent of the order of application of the rules.
- Evaluation of a mathematical expression is said to have the Church-Rosser Property:
- A simple example:
  - "If both ends of a string are consonants, remove one"



Church-Rosser Property

- How about expressions in programming languages? Do they have Church-Rosser Property?
- Answer it yourself considering these:
  - Limitations due to fixed size representations of numbers: Remember that a+(b+c) may not be equivalent to (a+b)+c?
  - Side-effects in evaluating some operations and function calls
    - f(2) + x

LESSON: A programmer has to know the order an expression is evaluated!

Side effect

```
1 def f(L):
2       L[0] += 2
3       return L[0]
4
5 M=[2, 3, 4]
6 x = f(M) + M[0]
```

Evaluation order yields two different results

Evaluate f(M) first

Evaluate M[0] first