Extended Introduction to Computer Science CS1001.py

Chapter A Lecture 4 — Python Memory Model (cont.), Collections, Expression in Python

Amir Rubinstein, Michal Kleinbort

School of Computer Science
Tel-Aviv University

Fall Semester 2023-24
http://tau-cs1001-py.wikidot.com

* Slides based on a course designed by Prof. Benny Chor

עדכונים קצרים

- מודל חסם את האפשרות להגיש קבצי py מסיבות אבטחה. נעדכן בקרוב כיצד להגיש. עדיין אין עדכון לגבי זה.
- ביום ראשון הקרוב תתקיים הפסקה לימודים בין 14-15 בה תתקיים עצרת להחזרת החטופים.
  - השיעור יתקיים בין 15:10-16:00.

Last Time

- List comprehension
- Functions
  - Definition, formal parameters
  - Call, actual parameters
  - return value
- "The three C's"
  - Correctness
  - Complexity
  - Clarity
- Python's Memory Model
  - Equality vs. identity
  - Mutable vs. immutable types, assignment vs. mutation
  - ~~Function call mechanism (by address)~~ today

This Lecture

- Python's Memory Model (cont.)
  - Function call mechanism (by address)
- More Collections Types in Python
  - tuple, dictionary, set
- Expression in Python

- Randomness

Python’s memory model

- Will be done mostly interactively in class. Slides that cover this are also available in the course site.
- The next slides summarize what we will see

Python's Memory Model (reminder)

- Objects are stored at specific memory locations

  `id(object1) == id(object2)` if and only if `object1 is object2`

- Warning: For optimization reasons, two objects with non-overlapping lifetimes may have the same id value. Furthermore, in two different executions, the same object may be assigned different id. And obviously this is platform dependent.

- Variables are temporary names for memory addresses

- Memory address does not imply value, Value does not imply memory address (except for “small” integers, and some strings, for optimization)

- Assignment of one variable to another merely creates another reference to the object.

- Mutable objects, such as lists, allow changing their "inner components" without changing the memory location of the "containing" object.

- Python Tutor http://www.pythontutor.com/visualize.html#mode=edit.

Python's Mechanism for Passing Functions' Parameters

- Different programming languages have different mechanisms for passing arguments when a function is called (executed).

- In Python, the address of the actual parameter is passed to the corresponding formal parameters in the function.

- An assignment to the formal parameter within the function body creates a new object, and causes the formal parameter to address it. This change is not visible to the original caller's environment.

- However, when the function execution mutates one of its parameters, its address in the function does not change, and it remains in the same address as in the calling environment. So such mutation does affect the original caller's environment. This phenomena is known as a side effect.

Information Flow and Side Effects of functions

- To conclude, we saw three ways of passing information from a function back to its original caller:

  1. Using `return` value(s). This typically is the safest and easiest to understand mechanism.
  2. Mutating a mutable formal parameter. This often is harder to understand and debug, and more error prone.
  3. Via changes to variables that are explicitly declared global. Again, often harder to understand and debug, and more error prone.

Comic Relief *

* אני מזמין אתכם לשלוח לי הצעות לתמונות שיופיעו על שקפים אלו לאורך הסמסטר

More Collections in Python

- As you recall, collections (aka containers) are objects that contain other “inner” elements.
  - We saw types `str`, `list`, `range`

- There are other useful collections in Python. Here are common ones, classified by two properties: order and by mutability.

	Ordered (sequence)		**unordered**	
	type	example	type	Example

Mutable	`list`	`[1,2,3]`	`set`
			`dict`
				`{1,2,3}`
				`{1:"a", 2:"b", 3:"c"}`

Immutable	`str`
	`range`
	`Tuple`
		`"123"`
		`range(1,4)`
		`(1,2,3)`		

Tuples

- Single, double, triple,… → tuple
- Tuples are much like lists, but syntactically they are enclosed in regular brackets, while lists are enclosed in square brackets.
- In contrast to lists, tuples are immutable

```
>>> a = (2,3,4)
>>> b = [2,3,4]
>>> type(a)
<class 'tuple'>
>>> type(b)
<class 'list'>
>>> [a[i]==b[i] for i in range(3)]
[True, True, True]
>>> a==b
False
>>> b[0] = 99
>>> a[0] = 99
TypeError: 'tuple' object does not support item assignment
```

Using tuples for function return Values

- Recall a function can return a single value. But look at this:

```
def mydivmod(a,b):
    """ return quotient and remainder """
    return a//b, a%b

>>> mydivmod(21,5)
(4, 1)
>>> type(mydivmod(21,5))
<class 'tuple'>
>>> mydivmod(21,5)[0]
4
>>> q,r = mydivmod(21,5)
>>> print(q,r)
4 1
```

Simultaneous assignment
(a convenient shortcut)

- Syntactically, this function returns a single value of type tuple. So practically, we can "bypass" the above mentioned constraint.

Sets (type `set`)

- Python’s `set` closely resemble the mathematical notion of a set
- No repetitions, no order
- Set members must be immutable (we may get back to this later in the course). Note the set itself is mutable.

```
>>> s = {1,2,3,"a"}
>>> type(s)
<class 'set'>
>>> s.add(4) # s is changed in-place, returns None
>>> s
{1, 2, 3, 4, 'a'}
>>> s.add("4")
>>> s
{1, 2, 3, 4, '4', 'a'}
>>> s.intersection({1,11,111})
{1}
>>> s.union({1,11,111})
{1, 2, 3, 4, '4', 11, 111, 'a'}
```

- We urge you to explore additional useful functionalities od sets

Dictionaries (type `dict`)

- Python’s `dict`s contain pairs of key:value. Used to represent mappings: a set of keys, each mapped to some value.
- Keys cannot repeat and are immutable (thus form a set). Note the dict itself is mutable.

```
>>> d = {"France":"Europe", "Germany":"Europe", "Japan":"Asia"}
>>> type(d)
<class 'dict'>
>>> d # order of elements not necessarily as in initialization
{'Germany':'Europe', 'France':'Europe', 'Japan':'Asia'}
>>> d["Japan"]
'Asia'
>>> d["Asia"]
KeyError: 'Asia'
>>> d["Israel"]
KeyError: 'Israel'
>>> d["Israel"] = "Asia"
>>> d["Israel"]
'Asia'
```

Dictionaries (type `dict`)

```
>>> d = {"France":"Europe", "Germany":"Europe", "Japan":"Asia"}

>>> for key in d:
     print(key, "is in", d[key])

Germany is in Europe
France is in Europe
Japan is in Asia
```

dict (and set) are iterable

- Note: order of elements not necessarily as in initialization
- This actually changed in Python version 3.7: Dictionary order is guaranteed to be insertion order. However, it's not a good practice to rely on it because it is version/language dependent.

Dictionaries (type `dict`) – Example

- Let's write a function that computes the number of occurrences of each letter in a given text.

- Input: `text` (type string)
- Output: pairs letter:count (type dict)

```
def char_count(text):
    d = {}
    for ch in text:
        if ch in d:
            d[ch] += 1
        else:
            d[ch] = 1
    return d
```

Advantages of `dict` and `set` over ordered collections

- Why should we consider using `dict` or `set` in the first place?
- Key observation, not explained at this point in the course (but it will be, when we see hash tables later on): membership checking is much "cheaper"
  - In particular, checking if an element belongs to a set or a dictionary is an operation whose efficiency does not depend on the size of the collection

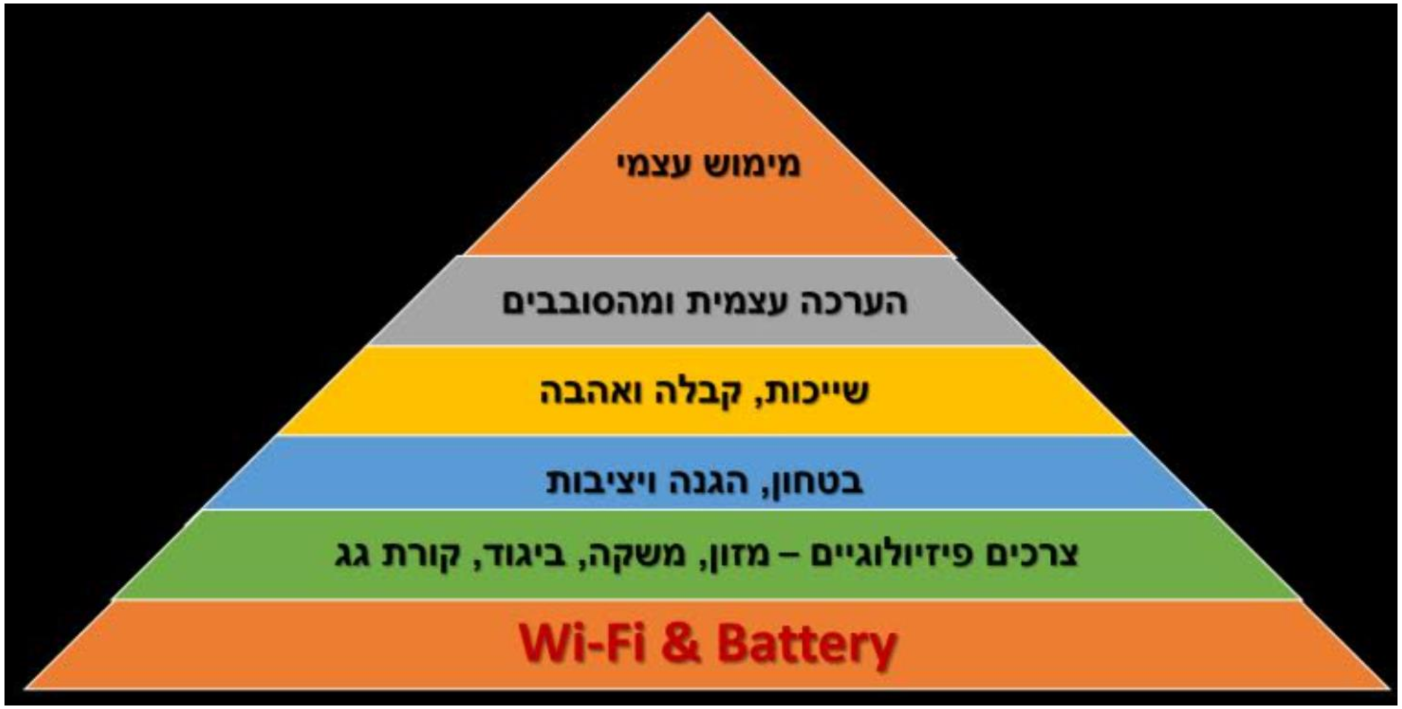
```
>>> s = {1,2,3,"a"}
>>> 2 in s
True
>>> "2" in s
False
```

A larger set will require roughly the same time

```
>>> lst = [1,2,3,"a"]
>>> 2 in lst
True
>>> "2" in lst
False
```

A larger list will require more time

Comic Relief *



* אני מזמין אתכם לשלוח לי הצעות לתמונות שיופיעו על שקפים אלו לאורך הסמסטר

Before we Move Beyond Python

- We are almost done with the ”introduction to Python” part pf our course

- The following slides contain material that will not be taught this semester about grammars, with one exception: a specific slide about expressions in Python

- All the slides are left here for reference

Grammars, Syntax vs. Semantics

- Syntax: the form of a valid program
  - Every language has its own syntax
  - Example: `print("abc")` is a valid form, `print("abc)` is not
  - Syntax can be defined by a grammar (next slides)

- Semantics: the meaning of the program and the expected results of executing it
  - Example: `print("abc")` will print the string inside the brackets to the screen

Specifying a Syntax

- The syntax of a programming language is formally defined using a Grammar
  - Similar to the case of Natural Language, yet with much less irregularities

- Reading Grammars takes some getting-used-to, but is not hard

- Before evaluating your program, the interpreter (e.g. IDLE) verifies that it conforms to the Grammar

Specifying Semantics?

- Much more cumbersome to do formally, and we will not cover this in this Intro course

- You will see a bit of that in the 3rd year Compilation course, and more if you study electives related to Software Verification

Grammar

- A grammar is defined by the following:
  - The alphabet of the language
  - A set of variables representing types of phrases or clauses in the sentence
  - The set of rules of the grammar

- We say a grammar forms (or yields) a string if we can derive the string using the rules repeatedly, until there are no variables left

Example #1

<SENTENCE> → <NP> <VERB>
<NP> → <ARTICLE> <NOUN>
<NOUN> → boy | girl | cat
<ARTICLE> → a | the
<VERB> → touches | likes | sees

| stands for "or"

Which strings can be formed using this grammar?

<SENTENCE> → <NP> <VERB>
→ <ARTICLE> <NOUN> <VERB>
→ a <NOUN> <VERB>
→ a girl <VERB>
→ a girl likes

□□

Example #1

<SENTENCE> → <NP> <VERB>

<NP> → <ARTICLE> <NOUN>

<NOUN> → boy | girl | cat

<ARTICLE> → a | the

<VERB> → touches | likes | sees

Which of the following strings can be formed using this grammar?

"a girl likes" ✓

"the boy sees" ✓

"the girl likes the cat" ✗

Example #2

$\langle S \rangle \rightarrow a\langle S \rangle a \mid b\langle S \rangle b \mid c$

Which strings can be formed using this grammar?

$\langle S \rangle \rightarrow a\langle S \rangle a$

$\rightarrow a\underline{a\langle S \rangle a}a$

$\rightarrow aa\underline{b\langle S \rangle b}aa$

$\rightarrow aab\underline{c}baa$

Example #2

⟨S⟩ → a⟨S⟩a | b⟨S⟩b | c

Which strings can be formed using this grammar?

⟨S⟩ → c

□

Example #2

$\langle S \rangle \to a\langle S \rangle a \mid b\langle S \rangle b \mid c$

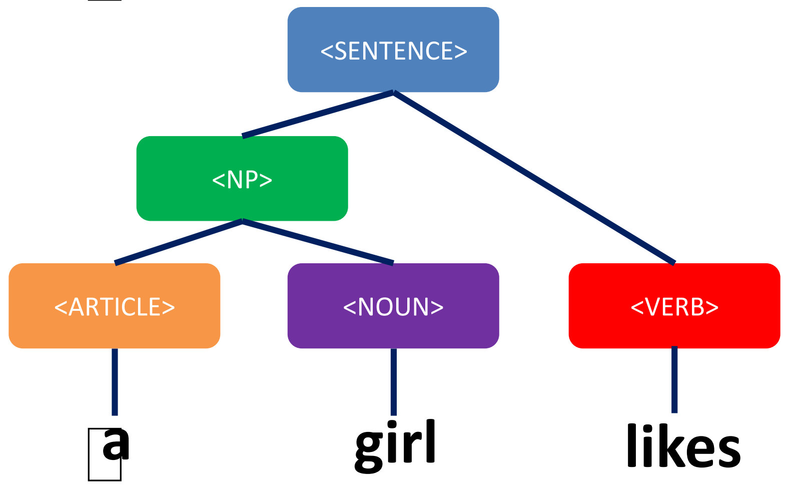
Can you generalize:

which strings can be formed using this grammar?

Parse Tree

- Parsing is the process of analyzing the syntactic structure of a string.
- This can be represented in the following form, termed parse tree or syntax tree.

Parse tree
for
“a girl likes”



Python's Grammar

- Python's full grammar is defined here
  - You are not required to understand what's written there, though

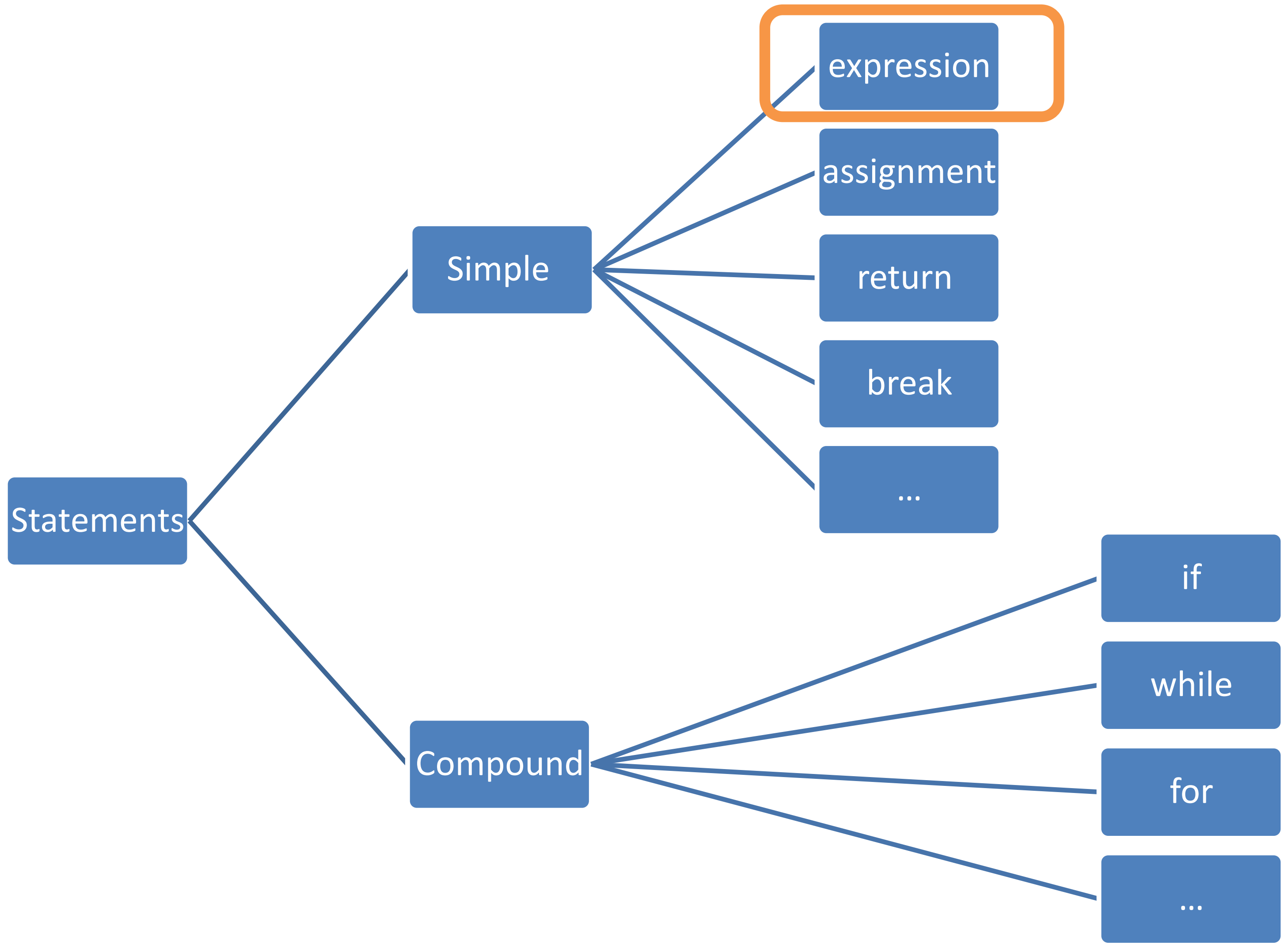
- Python code is parsed according to this grammar

Statements in Python

- Can be either simple or compound

Simple statements	**Compound statements**
• expressions, e.g., `3+4**2`	• if statement `if a > b:` `    condition block`
• assignments, e.g., `res = 400`	• while statement `while a > b:` `    loop block`
• return statement `return res`	• for statement `for a in lst:` `    loop block`
• break statement `break`	
and many more examples	and many more examples

Statements in Python: Sketch



Expressions

- An expression is a statement which "has a value". That is, anything that can be the right hand side of an assignment (e.g., `res=<expression>`).
- Examples:

```
res =   1
        3+2
        x #Suppose x was defined
        x>y
        x>y and "A" in "Amir"
        sum([1,2,3,4,5])
        [x**2 for x in [1,2,3] if x-1 != 0]
        None
```

Also:

```
sum
"equal" if x==4 else "not equal"
lambda x,y: x+y
```

Conditional expression

More about lambda expressions soon

מבנה ונושאי הקורס (באדום - חומר שירד בשל קיצור הסמסטר)

פרק	נושאים מתוכננים
A. יסודות פייתון	• תכנות בסיסי: טיפוסי ערכים, משתנים, משפטי תנאי, לולאות, פונקציות, מודל הזיכרון • נושאים נוספים: ~~דקדוקים פורמליים ותהליך הפירוש של פייתון~~, פונקציות למבדא, ופונקציות סדר גבוה, אקראיות ושימושיה, סוגי שגיאות (תחביר, זמן ריצה), סגנון תכנות "נכון"
B. ייצוג טיפוסי מידע	• ייצוג שלמים בשיטה הבינארית • ייצוג מספרים עם נקודה עשרונית בשיטת floating point • ייצוג תווים (ASCII, Unicode)
C. אלגוריתמים בסיסיים וסיבוכיות	• חיפוש בינארי, מיון בחירה, מיזוג רשימות ממוינות • סיבוכיות ו- O notation
D. חישוב נומרי	• מציאת שורש של פונקציה ממשית רציפה בשיטת החציה ~~בעבר: שיטת ניוטון-רפסון~~, חישוב נגזרות ואינטגרלים, קירוב ל π
E. רקורסיה	• עצרת, פיבונאצ'י, חיפוש בינארי, מיון מהיר, מיון מיזוג, ממואיזציה, דוגמאות נוספות
F. נושאים בתורת המספרים	• העלאה בחזקה טבעית בשיטת Iterated squaring • בדיקת ראשוניות הסתברותית (המשפט הקטן של פרמה) • פרוטוקול Diffie-Hellman להחלפת מפתח סודי • מחלק משותף מקסימלי (GCD)
G. תכנות מונחה עצמים (OOP) ומבני נתונים	• מחלקות, שדות ומתודות • רשימות מקושרות והשוואה לרשימות של פייתון • עצי חיפוש בינאריים • טבלאות hash • זרמים (streams) ופונקציות גנרטור
H. טקסט	• אלגוריתם CYK ~~בעבר: אלגוריתם קארפ-רבין~~ • דחיסת האפמן, דחיסת למפל זיו
I. ייצוג ועיבוד תמונה	• ייצוג תמונה דיגיטלית, ניקוי רעש (ממוצע וחציון מקומי), נושאים נוספים לפי הזמן
J. קודים לגילוי ולתיקון שגיאות	• ספרת ביקורת, קוד חזרה, ביט זוגיות, מרחק האמינג, קוד האמינג