http://www.cs.cornell.edu/courses/cs1110/2021sp

Lecture 2: Variables & Assignments

(Sections 2.1-2.3, 2.5, 2.6)

CS 1110

Introduction to Computing Using Python

Cornell CIS

COMPUTING AND INFORMATION SCIENCE

[E. Andersen, A. Bracy, D. Fan, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

http://www.cs.cornell.edu/courses/cs1110/2021sp

Lecture 2: Variables & Assignments

(Sections 2.1-2.3, 2.5, 2.6)

Have pencil and paper (or stylus and tablet) ready. We'll do visualization exercises that involve drawing diagrams today.

Recommendations for note taking:
- *Print out posted lecture slides and write on them*
- *Have the slides pdf ready and annotate electronically*

Lab 1 announcements

- Weren't able to attend lab? Don't panic. Do it on your own via link on course website. You all will get an extension on Lab 1 until Wednesday 17th
- To get credit in the online lab system you need this info:
  - For the short-answer in the boolean activity, include the term "short-circuit evaluation" for Python's behavior
  - Secret passwords for the 2 activities that ask for them:

    mod

    shortcircuit

More announcements

- Course website: http://www.cs.cornell.edu/courses/cs1110/2021sp/ Make sure it's spring 2021—look for the white cat logo  .
- Due to email volume, we can't answer emails to our personal addresses. If you mailed either prof at their individual email addresses but haven't yet got the info you need, please *post your question to Ed Discussions or use the email addresses listed on the "Staff" page.*
- Be sure to read/watch pre-lecture lessons before lecture. See "Schedule" page on website. Lecture assumes you have done the pre-lecture lessons.

Even more announcements

- Textbook is free online (link on website). **DO NOT CLICK** Instant Access on Canvas except to **OPT OUT**.
- CIS Partner Finding Social tonight 7:30-9pm. RSVP at **http://bit.ly/cisSP21**. Can't attend? Another good place to find a partner is your lab section. Talk with labmates!
- Install Anaconda Python 3.7 or 3.8 *and Atom editor* according to instructions on course website

Helping you succeed in this class

http://www.cs.cornell.edu/courses/cs1110/2021sp/staff/

Consulting Hours. Online with queuing
- Big block of time, multiple consultants (see staff calendar)
- Good for assignment help

TA Office Hours. Online
- Staff: 1 TA, 1 or two hours at a time (see staff calendar)
- Good for conceptual help

Prof Office Hours.
- After lecture for an hour. We'll try different tools to see what will work for us
- Prof. Fan has additional drop-in hours (see staff calendar)
- Prof. Lee has additional hours by appointment (use link on course website, *Staff/OH → Office Hours*)

Ed Discussions. Online forum to ask/answer questions (use link on course website)

AEW (ENGRG 1010). "Academic Excellence Workshops"
- *Optional* discussion course that runs parallel to this class. See website for more info

From last time: **Types**

Type: set of values & operations on them

Type **float:**
- Values: real numbers
- Ops: +, -, *, /,//,**

Type **int:**
- Values: integers
- Ops: +, -, *,/, //, %, **

Type **bool:**
- Values: `true`, `false`
- Ops: `not`, `and`, `or`

One more type today:

Type **str:**
- Values: string literals
  - Double quotes: **"abc"**
  - Single quotes: **'abc'**
- Ops: + (concatenation)

Type: `str` (string) for text

Values: any sequence of characters
Operation(s): **+** (catenation, or concatenation)
Notice: meaning of operator **+** changes from type to type

String literal: sequence of characters in quotes
- Double quotes: " abcex3$g<&" or "Hello World!"
- Single quotes: 'Hello World!'

Concatenation applies only to strings
- "ab" + "cd" evaluates to "abcd"
- "ab" + 2 produces an **error**

```
>>> terminal time >>>
```


Converting from one type to another

aka "casting"

<type>(*<value>*)

```
>>> float(2)
2.0

>>>int(2.6)
2


>>>type(2)
<class 'int'>
```

converts value 2 to type **float**

converts value 2.6 to type **int**

…different from:

type(*<value>*)

which tells you the type

What does Python do?

```
>>> 1/2.6
```

A. turn 2.6 into the integer 2, then calculate 1/2 → 0.5
B. turn 2.6 into the integer 2, then calculate 1//2 → 0
C. turn 1 into the float 1.0, then calculate 1.0/2.6 → 0.3846…
D. Produce a `TypeError` telling you it cannot do this.
E. Exit Python

Widening Conversion (OK!)

From a **narrower** type to a **wider** type
(e.g., `int` → `float`)

Width refers to information capacity. "Wide" → more information capacity

Python does it automatically if needed:
- Example: `1/2.0` evaluates to a `float`: `0.5`
- Example: `True + 1` evaluates to an `int`: `2`
  - `True` converts to `1`
  - `False` converts to `0`

From narrow to wide: **bool → int → float**

Note: does not work for **str**
- Example: `2 + "ab"` produces a TypeError

Narrowing Conversion (is it OK???)

From a **wider** type to a **narrower** type
(e.g., `float` → `int`)
- causes information to be lost
- Python **never** does this automatically

What about:

```
>>> 1/int(2.6)
```

Types matter!

You Decide:

- What is the right type for my data?
- When is the right time for conversion (if any)?

- Zip Code as an **`int`**?
- Grades as an **`int`**?
- Lab Grades as a **`bool`**?
- Interest level as **`bool`** or **`float`**?

Operator Precedence

What is the difference between:

2*(1+3) 2*1 + 3

add, then multiply ***multiply, then add***

Operations performed in a set order

- Parentheses make the order explicit

What if there are no parentheses?

→ **Operator Precedence:** fixed order to process operators when no parentheses

Precedence of Python Operators

- **Exponentiation**: **
- **Unary operators**: + –
- **Binary arithmetic**: * / %
- **Binary arithmetic**: + –
- **Comparisons**: < > <= >=
- **Equality relations**: == !=
- **Logical not**
- **Logical and**
- **Logical or**

- Precedence goes downwards
  - Parentheses highest
  - Logical ops lowest
- Same line → same precedence
  - Read "ties" left to right (except for **)
  - Example: 1/2*3 is (1/2)*3

- Section 2.5 in your text
- See website for more info
- Part of Lab 1

Operators and Type Conversions

Operator Precedence
Exponentiation: **
Unary operators: + –
Binary arithmetic: * / %
Binary arithmetic: + –
Comparisons: < > <= >=
Equality relations: == !=
Logical not
Logical and
Logical or

Evaluate this expression:

```
False + 1 + 3.0 / 3
```

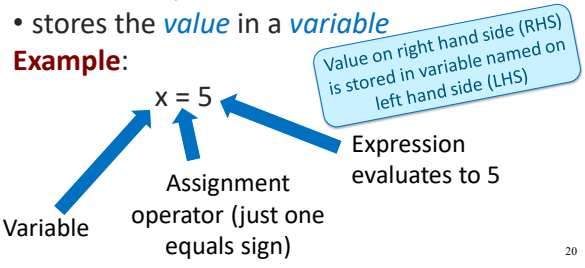
A. 3
B. 3.0
C. 1.3333
D. 2
E. 2.0

New Tool: Variable Assignment

An *assignment statement:*

- takes an *expression*
- evaluates it, and
- stores the *value* in a *variable*

Example:



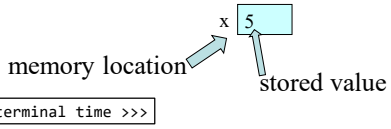
Executing Assignment Statements

```
>>> x = 5
>>>
```

Press ENTER and…

Hmm, looks like nothing happened…

- But something did happen!
- Python *assigned* the *value* 5 to the *variable* x
- Internally (and invisible to you):




```
>>> terminal time >>>
```

Retrieving Variables

```
>>> x = 5
>>> x
5
>>>
```

Press ENTER and…

Interactive mode tells me the value of x

`>>> terminal time >>>`

In More Detail: Variables (Section 2.1)

- A **variable**
  - is a **named** memory location (**box**)
  - contains a **value** (in the box)
- Examples:

The type belongs to the *value*, not to the *variable*.

Variable names must start with a letter (or _).

x	5	Variable **x**, with value 5 (of type **int**)
area	20.1	Variable **area**, w/ value 20.1 (of type **float**)

In More Detail: Statements

```
>>> x = 5
>>>
```

Press ENTER and…

Hm, looks like nothing happened…

- This is a **statement**, not an **expression**
  - Tells the computer to DO something (not give a value)
  - Typing it into >>> gets no response (but it is working)

Expressions vs. Statements

Expression	Statement
• **Represents** something ▪ Python *evaluates it* ▪ End result is a value	• **Does** something ▪ Python *executes it* ▪ Need not result in a value
• Examples: ▪ 2.3 (Value) ▪ (3+5)/4 (Complex Expression) ▪ x == 5	• Examples: ▪ x = 2 + 1 ▪ x = 5

Look so similar
but they are not!

You can assign more than literals

```
>>> x = 5
>>> x = 3.0 ** 2 + 4 – 1
>>> x = 2 + x
```

"x gets 5"

"x gets the value of this expression"

"x gets 2 plus the current value of x"

The RHS is an expression. An expression can include *literals*, *operators*, and *variables*.

Keeping Track of Variables

- Draw boxes on paper:

```
>>> x = 9
```

- New variable created?

```
>>> y = 3
```

Write a new box.

- Variable updated?

```
>>> x = 5
```

Cross out old value. Insert new value.

x	~~9~~ 5
y	3

Start with variable x having value 5. Draw it on paper:

x | 5

Task: Execute the statement `x = x + 2`

1. Evaluate the RHS expression, `x + 2`
   - For `x`, use the value in variable `x`
   - What value does the RHS expression evaluate to?
2. Store the value of the RHS expression in variable named on LHS, `x`
   - Cross off the old value in the box
   - Write the new value in the box for `x`

Execute the Statement: `x = 3.0*x+1.0`

Begin with this:

x | 7

1. Evaluate the expression `3.0*x+1.0`

2. Store its value in `x`

Executing an Assignment Statement

The command: `x = 3.0*x+1.0`

"Executing the command":

1. Evaluate right hand side `3.0*x+1.0`

2. Store the value in the variable `x`'s box

- Requires both evaluate AND store steps
- Critical mental model for learning Python

Exercise 1: Understanding Assignment

Have variable `x` already from previous

Create a new variable:

```
>>> rate = 4
```

x | 22.0

rate | 4

Execute this assignment:

```
>>> rate = x / rate
```

Dynamic Typing

Python is a **dynamically typed** language

- Variables can hold values of any type
- Variables can hold different types at different times

The following is acceptable in Python:

```
>>> x = 1          ← x contains an int value
>>> x = x / 2.0    ← x now contains a float value
```

Alternative: a **statically typed** language

- Examples: Java, C
- Each variable restricted to values of just one type

Exercise 2: Understanding Assignment

Begin with:

x	22.0
rate	5.5

Execute this assignment:

```
>>> rat = x + rate
```

More Detail: Testing Types

May want to track the type in a variable
Command: `type(<expression>)`

Can get the type of a variable:

```
>>> x = 5
>>> type(x)
<class 'int'>
```

Can test a type with a Boolean expression:

```
>>> type(2) == int
True
```