Lecture 2: Variables & Assignments
(2.1-2.3, 2.5, 2.6 or videos (see Schedule))

CS 1110
Introduction to Computing Using Python

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

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

There were many questions about what certain operators do (/, //, %). **You do not need to memorize their behavior.** We want you to know about them so that when the need arises, you can make use of them.

Similarly, we want you to know that operator precedence **exists** so you can understand how Python works. Instead of memorizing these slides, you can reference the exact ordering when it matters to the code you are writing.

Lab 1 Activities

**Activity 1:** if you aren’t passing the "banner" question, get help with your installation! **In the meantime,** you can add the following string to your answer to get the system to accept it:

    Python 3.x Anaconda

**Activity 2:** the password is:

    learn.by.testing.hypotheses

(no spaces, all one "word", period separators)

**Activity 3:**

**Q3:** add the name of Python's behavior to your answer:

    short circuit evaluation

**Q8:** password:

    shortcircuit (no spaces)

Which of the following is **false?**

A **type**…

(a) is a set of values & operations on these values
(b) represents something
(c) can be determined by using type() in Python
(d) can be changed by using type() in Python
(e) determines the meaning of an operation

**If there are multiple false answers, pick one!**
From last time: **Types**

**Type: set of values & operations on them**

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

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

- **bool**: 
  - Values: True, False
  - Ops: not, and, or

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

Converting from one type to another
aka "casting"

```python
>>> float(2)
2.0
>>> int(2.6)
2
>>> type(2)
<class 'int'>
```

...different from:

```python
>>> type(<value>)
which tells you the type
```

What does Python do?

```python
>>> 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)

Python does automatically if needed:

- Example: `1/2.0` evaluates to a float: `0.5`

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:

```python
>>> 1/int(2.6)
0.5
```

*Python casts the 2.6 to 2 but / is a float division, so Python casts 1 to 1.0 and 2 to 2.0*

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:

\[ 2*(1+3) \]
\[ 2*1 + 3 \]

\textit{add, then multiply} \quad \textit{multiply, then add}

Operations performed in a set order
- Parentheses make the order explicit

What if there are no parentheses?
→ \textbf{Operator Precedence}: fixed order to process operators when no parentheses

Precedence of Python Operators

- Exponentiation: **
- Negation: -
- 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

Evaluate this expression:
\[ 7 + 3.0 / 3 \]

A. 3
B. 3.0
C. 3.333333335
D. 8
E. 8.0

New Tool: Variable Assignment

\textbf{Example}:
\[ x = 4 + 1 \]

An \textit{assignment statement}:
- takes an \textit{expression}
- evaluates it, and
- stores the \textit{value} in a \textit{variable}

Executing Assignment Statements

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

Press ENTER and…

Hmm, looks like nothing happened…

- But something did happen!
- Python \textit{assigned} the \textit{value} 5 to the \textit{variable} \texttt{x}
- Internally (and invisible to you):

  ```python
  x 5
  ```

  memory location

  stored value
Retrieving Variables in Interactive Mode

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

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**:
  - `x` is a variable, with value 5 (of type `int`)
  - `area` is a variable, with value 20.1 (of type `float`)

Variable names must start with a letter (or `_`). The type belongs to the value, not to the variable.

In More Detail: Statements

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

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**
  - Represents something
  - Python *evaluates it*
  - End result is a value

  - **Examples**:
    - `2.3`
    - `(3+6)/4`
    - `x == 5`

- **Statement**
  - Does something
  - Python *executes it*
  - Need not result in a value

  - **Examples**:
    - `x = 2 + 1`
    - `x = 5`

Look so similar but they are not!

Keeping Track of Variables

- **Draw boxes on paper**:
  - >>> `x = 5`

- **New variable declared?**
  - >>> `y = 3`
  - Write a new box.

- **Variable updated?**
  - >>> `x = 7`
  - Cross out old value. Insert new value.

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

```
  x 5
  y 3
```

**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 `x`
   - in variable names on LHS, `x`
   - Cross off the old value in the box
   - Write the new value in the box for `x`
Begin with this:

\[ x = x + 2 \]

1. Evaluate the expression \( 3.0 \times x + 1.0 \)
2. Store its value in \( x \)

Executing an Assignment Statement

The command: \( x = 3.0 \times x + 1.0 \)

"Executing the command":
1. Evaluate right hand side \( 3.0 \times x + 1.0 \)
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:

\[ \text{rate} = 4 \]

Execute this assignment:

\[ \text{rate} = x / \text{rate} \]
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:
```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:
```
\[
\begin{array}{c|c}
\hline
x & 22.0 \\
\hline
\end{array}
\begin{array}{c|c}
\hline
\text{rate} & 5.5 \\
\hline
\end{array}
\]
```

Execute this assignment:
```python
>>> rat = x + rate
```

Did you do the same thing as your neighbor? If not, **discuss**.

---

**Which one is closest to your answer?**

- A. \[
\begin{array}{c|c|c}
\hline
x & 22.0 & 27.5 \\
\hline
\text{rate} & 5.5 \\
\hline
\end{array}
\]
- B. \[
\begin{array}{c|c|c}
\hline
x & 22.0 \\
\hline
\text{rate} & 5.5 \\
\text{rat} & 27.5 \\
\hline
\end{array}
\]
- C. \[
\begin{array}{c|c|c}
\hline
x & 22.0 \\
\hline
\text{rate} & 5.5 \\
\text{rat} & 27.5 \\
\hline
\end{array}
\]
- D. \[
\begin{array}{c|c|c}
\hline
x & 22.0 & 27.5 \\
\hline
\text{rate} & 5.5 \\
\text{rat} & 27.5 \\
\hline
\end{array}
\]
- E. ekyll命

```
rat = x + rate
```

---

**More Detail: Testing Types**

**Command:** `type(<value>)`

Can test a variable:
```python
>>> x = 5
>>> type(x)
<class 'int'>
```

Can test a type with a Boolean expression:
```python
>>> type(2) == int
True
```