CS2110 Lecture 3 Jan. 29, 2021

ANNOUNCEMENTS

- First DS assignment due Feb. 2. You don't have to attend the Tuesday DS sections if you don't want to.
- First homework due Thurs. Feb. 4
- Nobody has posted a question on Piazza yet. And some of you have not "joined" Piazza. Would be good to see one so we can be sure it's working ☺

IMPORTANT THINGS TO DO THIS WEEK

- Read the textbook, Ch 1 and Ch2
- Install Python on your computer
- Practice basic expressions in the Python interpreter
- Make sure you can access Piazza (discussion forum) through the ICON

FOR MONDAY

- Start reading and doing exercises in Chapter 6 (Functions)

LAST TIME

- Course overview
- Chapter 1 of text
  - Starting Python and intro to evaluating simple expressions
  - **expressions** yield **values**
  - every value has a **type**
- IC Arrest blotter demo program

TODAY

Chapter 1.13

- Comments (#)

Chapter 2

- More detail on expressions
- Variables and assignment statements
- Strings and expressions

(from last time) Ch 1: Programs

Key components of programming, independent of particular programming language.

- Expressions
- Variables and assignment
- if-then-else (decision making/branching)
- Iteration
- Functions

Essentially all programming languages are built around these components. Once you understand how to describe computations using them, you can program. **Learn these basics well!** Changing programming languages is usually just a matter of looking up details of how to "say" a particular standard thing in the different language.

Chapter 1 of text says it differently: input, output, math, conditional execution, repetition. I/O is important but, to me, not key or interesting at the beginning. "math" is too vague. Assignment and variables, in the programming sense, don't fit well under everyday use of word "math." And while functions can be thought of as math, in programming functions (often called procedures when used slightly differently) play a very important role beyond just "being part of math." They help organize programs into understandable components, etc.

(from last time) Ch 1: expressions, arithmetic, types

- You can use the Python interpreter like a calculator, typing in many different mathematical (and other) expressions and seeing immediate results
  - print("hello")
  - 3 + 2
  - 2**128
  - 5/2+1
- Expressions yield **values.** Every value has a **type.**
  - 3's type in **int** (for integer)
  - 3.5's type is **float** (for floating point number)
    - (What about 3.0? 3.0's type is float. It is not the exact same thing as 3 in Python but (fortunately) it *is* "equal" to 3. More on this later.)
  - "hello"'s type is **string**
  - You can ask Python for a value's type

    >>> type(3.5)

Ch 2 - Expressions

Generally, an **expression** is a combination of literals (things like numbers, strings, Booleans) and operators (+, -, …) that, when evaluated by Python, yield a **value**

- **mathematical expressions**
  - yield numbers, objects of type **int** or **float**

```
3 * (200 / 1.5)
abs(-4) + 2
(2 * (5 // 2)) + (5 % 2)
```

  - Several mathematical operators are discussed Sec 2.7
- **logical expressions**
  - yield **True** or **False,** objects of type **bool**

Logical expressions

Some logical operators (see Ch 7.1 and 7.2):

and, **or**, **not**, **>**, **<**, **==**

Example expressions:

```
>>> (1 < 3) and (0 > 2)
False
>>> abs(-1) == 2
False
>>> (5 == (2 + 3)) or True
True
```

Important notes:

- == is not a statement of equality! It's a question – are the two sides equal? True or False question!
- True and False are NOT strings. They are basic Python values different than "True" and "False". *Many students forget this*

Order of operations

The textbook has a section (2.9) on order of operations, so that you can figure out how to calculate the value of something like:

>>> 3 + 1 or 3 + 2 ** 2 + -14 / 2.0 == 0

???

This has a legal value but I wouldn't be able to tell you without looking things up. I **always** parenthesize fully to make expression clear.

E.g. ((3 * x) + (4.2 * (z ** 1.2))) - y

Code should be written so that you and others can read and understand it without working too hard!

Variables (2.4) and Assignment statements

Expressions yield values and we often want to give names to those values so we can use them in further calculations. A **variable** is a name associated with a value.

The **statement**

```
>>> x = 10
>>>
```

creates the variable x and associates it with value 10.

'x = 10' is a statement not an expression. It doesn't have or produce a value. BUT it does have an important effect - it associates x with the value 10 and subsequently x can be used in expressions!

```
>>> x + 3
13
```

Variables and Assignment Statements

In general, you write:

```
>>> var_name = expression
```

where var_name is a legal variable name (see book/Python reference to know what's legal) and expression is any expression

```
>>> zxy1213 = 14.3 + (3 * math.sin(math.pi/2))
>>> zxy1213
 17.3
```

And since zxy1213 is a variable, thus a legal expression, we can write:

```
>>> sillyVarName = zxy1213 – 1.0
>>> sillyVarName
16.3
```

Variables and Assignment Statements

Only variable names, not expressions, can appear on to the left of an = sign (unlike for ==, the equality "question")

>>> x = 3 + 4 OKAY

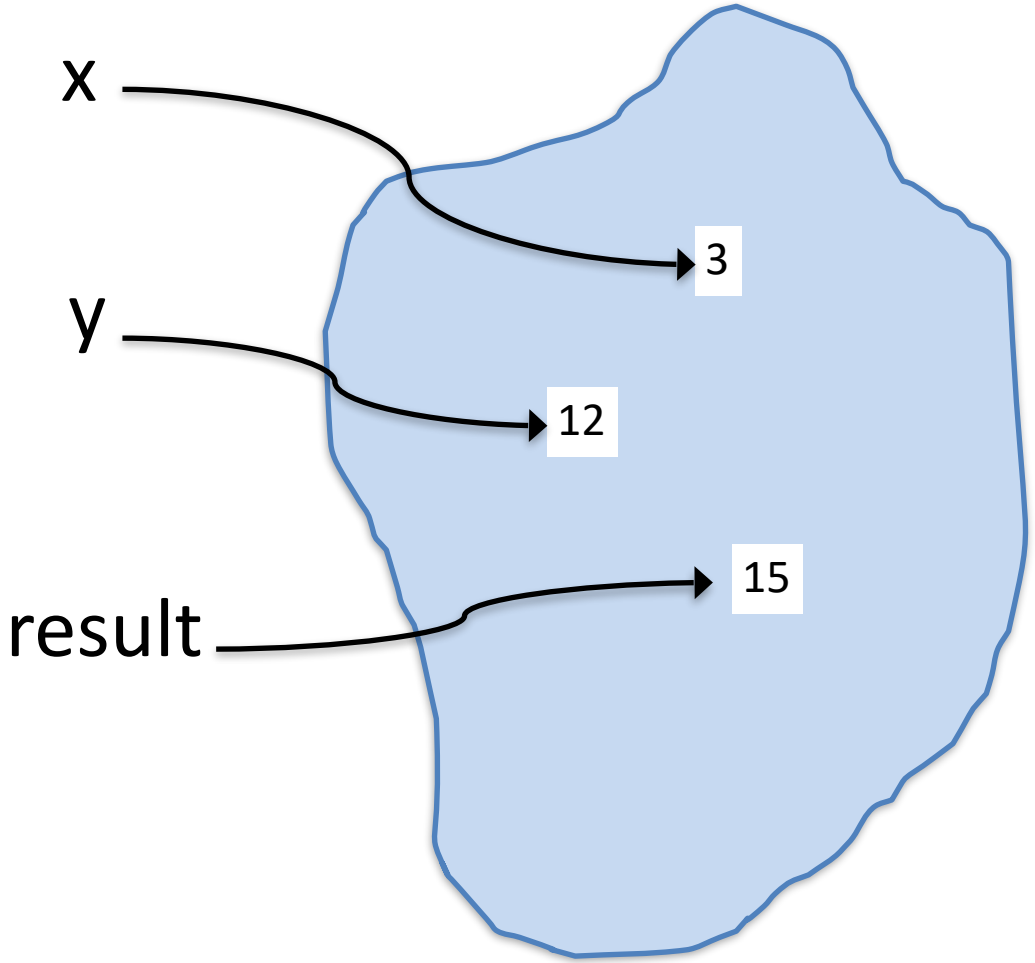
>>> x + 3 = 4 NOT OKAY (crashes, yields syntax error.)

>>> 3 = x + 4 NOT OKAY (crashes, yields syntax error.)

>>> x + 3 == 4 OK (will return True or False, or give error if x has not been assigned a value)

Variables and Assignment Statements

```
>>> x = 3
>>> y = 4 * x
>>> result = x + y
```



One of the most important things to know all semester

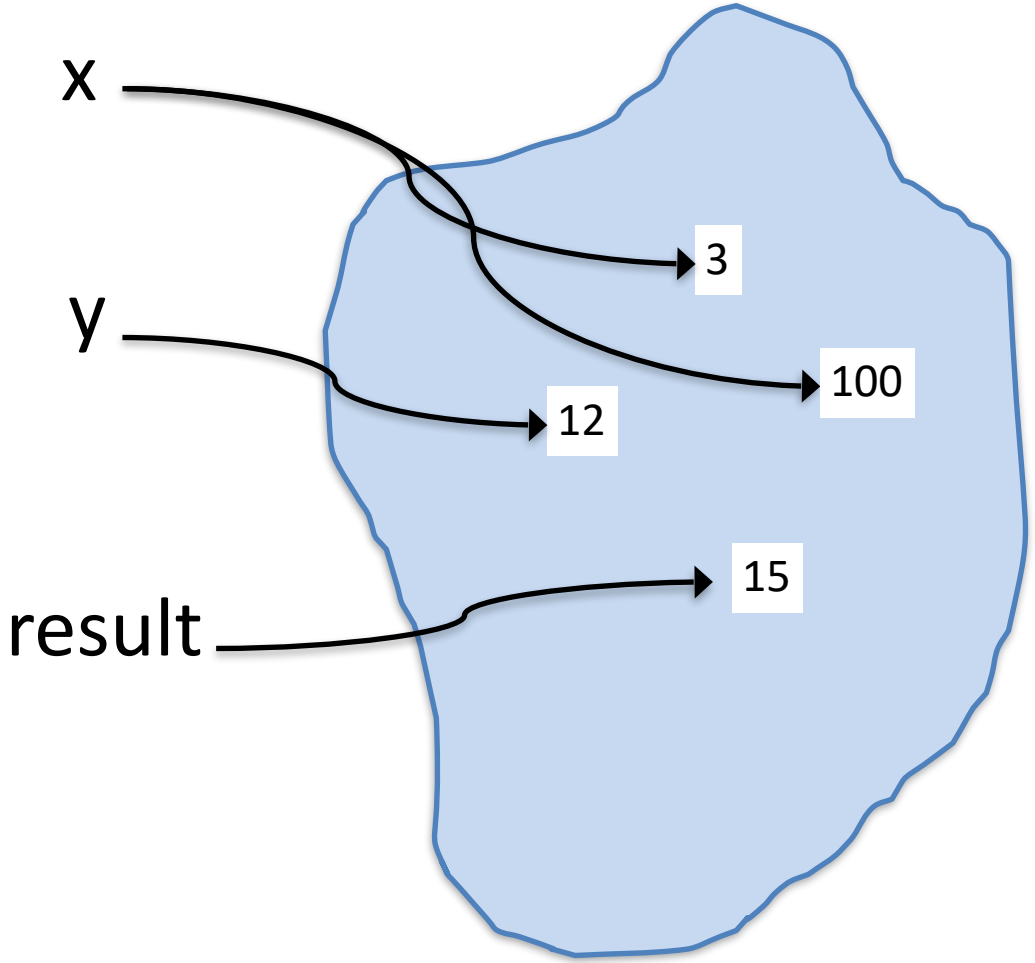
To process an assignment statement

x = y * 32.1 ** foo(...) – math.sin(bar(....))) / ...

1) Evaluate right hand side (ignore left for a moment!) yielding a value (**no variable** involved in result – it's a number or a string or a boolean value or …)
2) Associate variable name on left hand side with resulting value

Variables and Assignment Statements

```
>>> x = 3
>>> y = 4 * x
>>> result = x + y
```



Rule (*very important to remember*):

1) Evaluate right hand side (ignore left for a moment!) yielding a value (no variable involved in result)
2) Associate variable name on left hand side with resulting value

```
>>> x = 100
>>> y
?
>>>result
?
```

y and result are not changed!

Don't think of assignments as constraints or lasting algebraic equalities. They make (perhaps temporary) associations between names and values.

Ch2.2's information on strings

There's a whole chapter (9) on strings but we want to use them much sooner (for one thing, we need them to print nice output!)

- Use quotes (single, double, even triple!) to delimit:
  "abc" == 'abc'
- But "abc' is not a legal string.
- The quotes at the ends are *not* part of the string. "abc123" has 6 characters in it, not 8.
- Strings *can* contain quote characters E.g. "abc'def" is a 7 character string containing a, b, c, d, e, f, and a single-quote character
- Strings in Python are powerful and can get complicated. Can represent full Unicode, emoji, chars from many languages, …

Strings

- Again, we'll talk more about strings later but for now you should at least know you can use + operator (see 9.2) on them

```
>>> name = "Jim"
>>> sentence = "Hi " + name + "!"
>>> sentence
>>> "Hi Jim!"
>>> print(sentence)
Hi Jim!
>>>
```

- Also it's easy and useful to be able to convert numbers to strings

```
>>> "August has " + 31 + " days."
TypeError: can only concatenate str (not "int") to str
>>> "August has " + str(31) + " days."
'August has 31 days.'
```

- Ch 2.8 presents the “input” function that can be used to prompt users for values. It’s worth knowing but not key to most of the homework you’ll do. We will only use it a couple of times in special situations. Unless it’s clearly specified as part of a given homework or discussion assignment, you should *not* use the input function in your code for this class.

Side note: be careful with floats!

```
>>> v1 = 2 ** 64
>>> v2 = 2.0 ** 64
>>> v1 == v2
True
>>> (v1-1) == (v2-1)
False
>>> (2.0 **64) == (2.0**64) + 1
True
>>> (2.0**53) + 1 + 1 == (2.0**53) + 2
False
```

Those might seem esoteric but consider:

```
>>> (2.2 * 3) == 6.6
False
>>> .1 + .1  + .1 == .3
False
```

IMPORTANT LESSON: Floating point representations are often not exact. It probably won't be an issue in this class, but super important to keep in mind for the future. Be careful when comparing floating point numbers for equality with each other or with 0.0!

Instead of

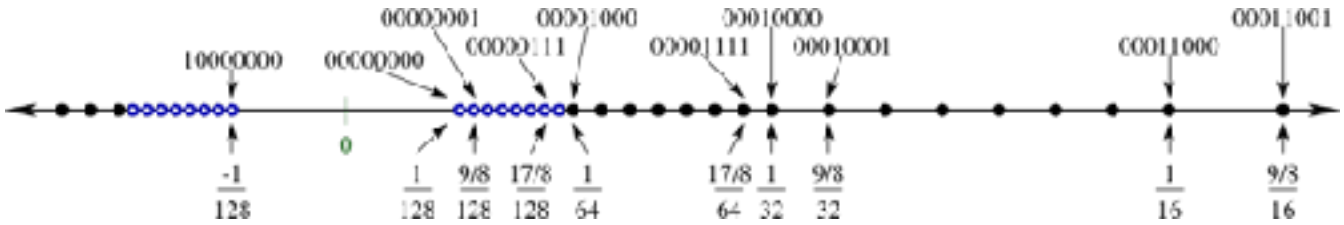
x == 0.0

use comparisons like

x <= abs(epsilon)

for some suitably small value epsilon

Floating point numbers are stored in computer with a fixed number of bits (e.g. 64) and thus can't possibly exactly represent all real values (not even all real values less than a certain value, because there is always another real value between any two)



Next time

- More on variables and assignment
- Ch 6: Functions
- For next time (and for disc. section), start reading and doing exercises in Ch6