CS1210 Lecture 3 Aug. 27, 2021

ANNOUNCEMENTS

- DS 1 assignment posted, due Aug 31, 8pm. You don't have to attend the Tuesday DS sections if you don't want to.
- Homework 1 due Friday, Sep 3, 2pm

IMPORTANT THINGS TO DO

- 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

- Reading and do exercises in Chapter 6 (Functions) - necessary for DS1 and HW1 assignments!

LAST TIME

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

TODAY

- Chapter 2
  - More detail on expressions
  - Variables and assignment statements
  - Strings and expressions
- DS1 assignment overview
- Initial intro to functions - Ch 6

(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. Type mathematical and other **expressions** and see immediate results

```
>>> print("hello")
Hello
>>> 3 + 2
5
>>> (3 > 2)
True
```

- 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)
<class 'float'>
```

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**

  3 > 2

  (len("hello") > 3) or (5 < a)

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 ==, the equality "question")

>>> x = 3 + 4 OKAY

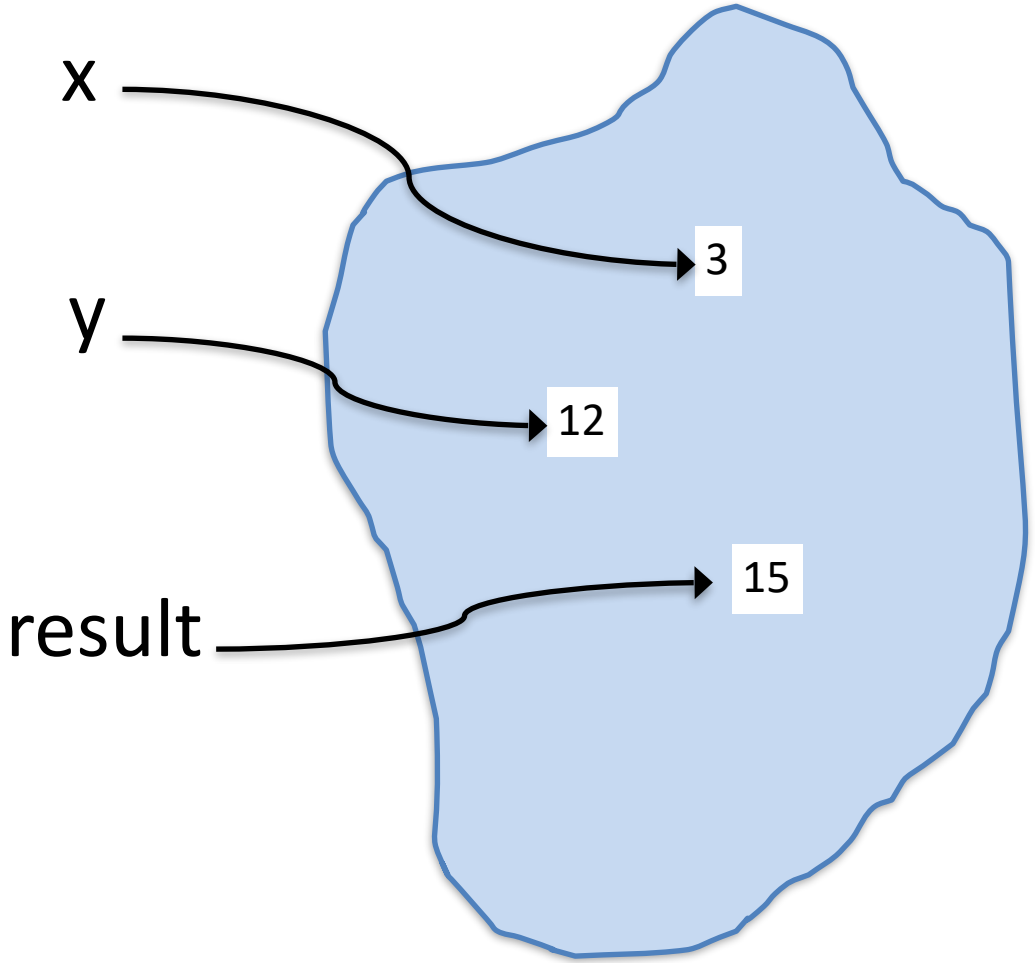
>>> x + 3 = 4 NOT OKAY (crashes, yields syntax error.)

>>> 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

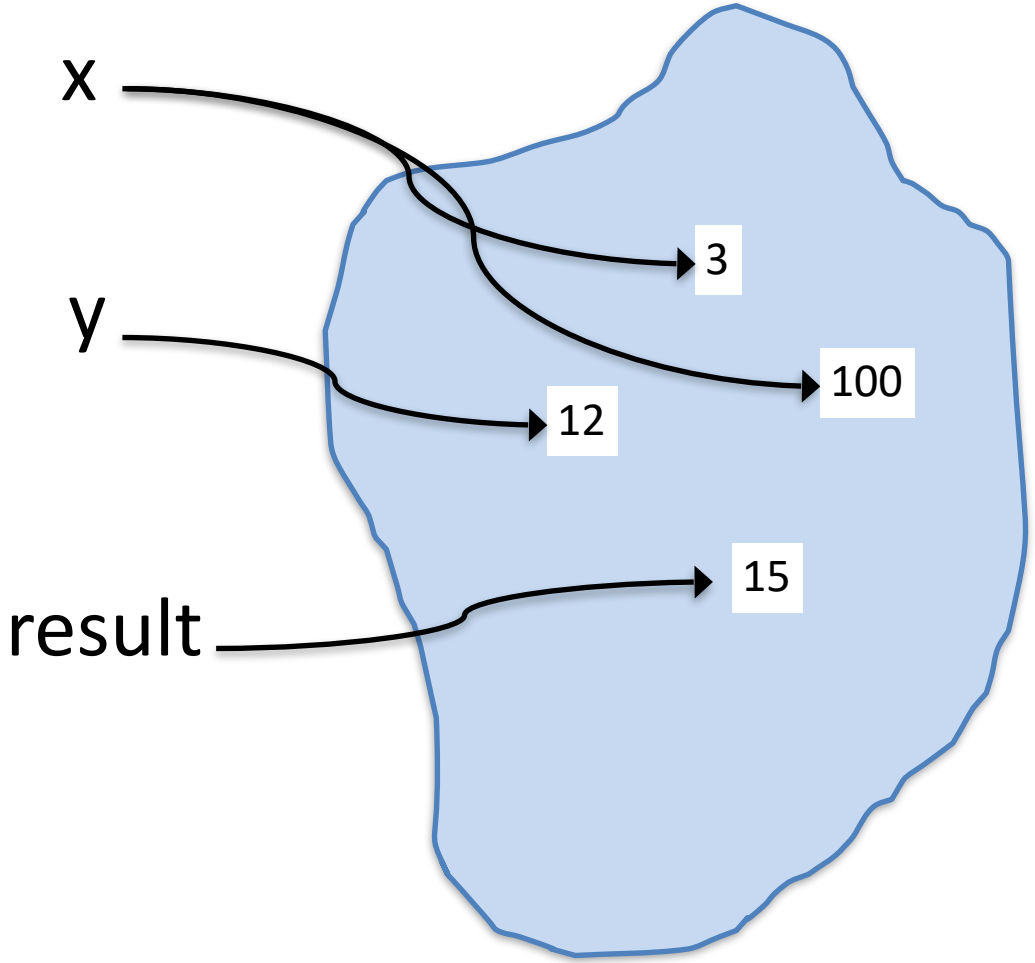
To process an assignment statement

result = 3.4 * x - math.sqrt(f(y*y)) - math.sin(g(z/4.1))

1) Evaluate right hand side (ignore left for a moment!) yielding a value (**no variables** involved in result – it's simply a number, string, boolean or other **value**)
2) Associate variable name on left hand side with the 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 make strings:

"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, …

```
>>> myString = "👍👀🐤"
```

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 using the built-in str function

```
>>> "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.

DS1 Assignment and introduction to Functions (Ch 6)

https://homepage.cs.uiowa.edu/~cremer/courses/cs1210/ds/ds1.py

- As with all assignments, you must implement **functions**
  - Textbook: "In Python, a **function** is a named sequence of statements that belong together. Their primary purpose is to help us organize programs into computational chunks that match how we think about the solution to a problem."
- We'll do a super quick introduction to functions today. More Monday
- In DS1, the three Part 3 functions are all closely related to work you have to do for HW1

Ch 6: Function definition and function calls

A **function call** is an expression. We say a function takes N argument values, executes the statements in the function definition, and **returns** a computed value, returned_value

```
>>> fn_name(arg1, arg2, …, argN)
returned_value
```

Function call examples.

```
>>> abs(-3)   ← function call
3             ← value returned from function call
>>> min(17, 4)   ← function call
4                ← value returned
```

abs and min are “built-in” functions, provided for us by Python.

A super important key component of programming is defining NEW functions of your own design via Python’s “def” statement

Ch 6: Defining New Functions

Super important to understand this! (You will do a *lot* of this in this course!)

Again, a function **call**, f(a,b,c) is an expression with a value, just like other Python expressions. Like in math, a function takes some "input" *arguments*, computes something, and *returns* an answer

def enables you to *define your own functions*

Ch 6: Defining New Functions

def functionName (param1, param2, …, paramN):

    ….
    …. (body of function, can be many lines,
    …. computes result value in terms of parameter
    …. variables bound to input values)
    ….
    return result_value

Make sure you understand:

- A primary use of functions is to define a general procedure:
  - Compute the square root of any non-negative number
  - Compute the minimum of a pair of number
  - Convert from a temperature in Celsius to Fahrenheit
- Computation in the function's body is specified in terms of the parameter variables (param1, …, paramN). The parameter variables will be bound to argument values when the function is called. (*We'll take much more on this over the next few days*)

Ch6: Defining functions

```
def myMin (a,b):
    if (a < b):
        return a
    else:
        return b
```

```
>>> myMin(5,7)
5
```

think of evaluating myMin(5,7) as executing:

```
a, b = 5, 7
if (a < b):
    return a
else:
    return b
```

5

Super important: Parameter variables a and b are only defined *during the execution of myMin*

```
>>> a
```

ERROR: a not defined

Ch 6: Defining functions

```
def myMin (a,b):
    if (a < b):
        return a
    else:
        return b
```

```
>> x, y = 12, 10
>> myMin(x,y)
```

myMin(12,10)

```
a, b = 12, 10
if (a < b):
    return a
else:
    return b
```

```
>> 10
```

Ch 6: Defining New Functions

```
def foo(a, b, c):
    temp = a * b
    result = temp + c
    return result
```

IMPORTANT

When executing a function call:

1) first, the function's parameter variables are bound to the *values* of the function call's arguments

2) second, the body of the function is executed

```
>>> x = 3
>>> foo(x * x, x + 1, 3)
```

← foo will be executed with variable

a bound to 9

b bound to 4

c bound to 3

foo "knows" nothing about x. x *isn't* passed in. 9, 4, and 3 are passed into foo.

Next time

More on:

- variables
- assignment
- functions (Ch 6)

For next time: read and do exercises in Ch6