

CS 106

Introduction to Computer Science I

07 / 01 / 2026

Instructor: Michael Eckmann

Today's Topics

- Recap
 - Functions: print, input, int, float
 - Variables and value types (int, str, float, bool)
 - Operators:
 - = (assignment),
 - + (concatenation or addition),
 - / (division)
 - // (integer division)
- Other arithmetic operators: -, %, *, **, ()
- If else statements

Recap

- Variables hold one value at a time.
- Values have a type
e.g. str, int, float, bool
- To assign a value to a variable the variable name must be on the left hand side of the assignment statement.

- e.g.

$x = 5$

$y = 7$

$x = y$ # which of these 2 variables gets a new value?

Recap

- Operators
- `=` is assignment operator
- `+` concatenates two str's together
- `+` is addition when used on numeric values
- `/` is floating point division
- `//` is integer division

Recap

- Printing: `print` is a function that takes in any number of arguments, separated by commas. `sep` and `end` are two parameters and we can change their default values
- Keyboard input: `input` is a function that takes a `str` as an argument (which is output to the screen) and waits for user input until enter key is pressed and returns what was typed as a `str`
- Converting `str` to `int`: `int` is a function that takes a `str` as an argument and returns the `int` version of that string

Examples

```
user_name = "Jane"  
print("Hello", user_name)
```

```
user_age = 21  
print(user_name, "will be", user_age + 10,  
      "years old in 2036")
```

Input example

```
height = input("How many feet tall are you?")
```

```
height = int(height)
```

```
# above line converts the str value to int
```

```
print("there are", height*12, "inches in",  
      height, "feet.")
```

Program Flow

- Python programs start executing at the top and they execute sequentially, in order (unless something changes the program flow --- more on that later today and throughout the semester) until the end of the source code file which is when the program ends running.

Arithmetic operators

We already used + and /

- is subtraction

* is multiplication

// is division resulting in an int

(whereas / is division resulting in a float)

% is modulus (remainder)

** is exponentiation

All work on two operands (one on the left of the operator and one on the right)

Operands could be variables OR values

The modulus operator (remainder)

- `%` is the modulus operator

`val1=16`

`val2 = 5`

`remain = val1 % val2`

- The modulus operator results in the remainder after **`val1`** is divided by **`val2`**.
- **`remain`** would have the value 1 because `16//5` is 3 with a remainder of 1
- `val1//val2` results in 3 (because `//` is int division)
- `val1/val2` results in 3.2 (`/` is float division)

Arithmetic Operator precedence levels

- () parentheses are evaluated first
- if parens are nested then the innermost pair is evaluated first.
- *, /, //, % multiplication, division and modulus are next --- if several of these, they are evaluated **left to right**.
- +, - addition and subtraction are last --- if several of these, they are evaluated **left to right**
- It's useful to memorize this precedence list

Arithmetic Operator precedence example

some_result = (5 + 4 - 1) * 6 + (2 - 10)

- *(correct way)*
- $5+4-1 \rightarrow 8,$
- $2-10 \rightarrow -8,$
- $8*6 \rightarrow 48,$
- $48-8 \rightarrow 40$

Equality and Relational operators

`==` is equal to operator

`!=` is not equal to operator

`<` is less than operator

`>` is greater than operator

`<=` is less than or equal to operator

`>=` is greater than or equal to operator

if/else statements

An if/else statement syntax is

```
if bool_expr:
```

```
    # do something if bool_expr is True
```

```
    # do something else if bool_expr is True
```

```
else:
```

```
    # do something if bool_expr is False
```

```
    # do something else if bool_expr is False
```

Notice the colons that is part of the syntax of if/else statements. Also, the code inside if and the else portions are indented. Indentation matters in python

else is optional

1 or more statements can be inside the if or the else portion

if / else

- `bool_expr` is something that will evaluate to True or False (usually using relational or equality operators.)
- After the if, one can have arbitrary many `elif bool_expr:` portions.
- Let's write some examples and use some relational and equality operators
 - An if alone
 - An if and an else
 - An if, elif, elif, else
 - An if, elif, elif

Do lab work now

- You should now do lab 01

Pseudocode

- pseudocode is an informal use of English to describe what a program is to do and in what order
- pseudocode is not an actual computer programming language
- it is used prior to writing actual code to help the programmer in the planning stages of a program

Example Application Exercise

- write a program to compute the number of projected home runs a baseball player will hit for the season based on how many homers he's hit so far.
- Output should look like:

player's name is projected to hit *number* home runs in 162 games.

- Any ideas?

Pseudocode for our example

get player's name from the user

get the number of homeruns so far

get the number of games played so far

compute the number of projected homeruns for this player
based on a season of 162 games by using the following
calculation

$$\frac{\text{projected homers}}{162} = \frac{\text{homers so far}}{\text{games played so far}}$$

Pseudocode for our example (continued)

from this equation,

$$\frac{\text{projected homers}}{162} = \frac{\text{homers so far}}{\text{games played so far}}$$

we can multiply both sides of the equation by 162 and get

$$\text{projected homers} = \text{homers so far} * 162 / \text{games played so far}$$

Pseudocode for our example

(continued)

Print out the following with actual values for *player's name* and *number*

player's name is projected to hit *number* home runs in 162 games.

Pseudocode could be more fleshed out than what we have done here --- use as much or as little detail in pseudocode as you prefer.

Pseudocode for our example

(continued)

- Now we can write the program based on our pseudocode.

More assignment operators besides =

- $+=$, $-=$, $*=$, $/=$, $\%=$
- Here's what they mean. Suppose we have:
 $a=7$
 $a = a + 6$ # is equivalent to: $a += 6$
- These assignment operators are used only if the variable on LHS (left hand side of the assignment) is also the first variable on the RHS.

More assignment operators besides =

- +=, -=, *=, /=, %=

More examples:

```
degrees -= 5
```

```
# subtract 5 from degrees and store this new value
```

```
# in degrees.
```

```
halve_me /= 2
```

```
# divides the value in halve_me by 2 and stores
```

```
# this new value in itself */
```

```
fine *= 3  # value in fine is tripled
```

Clarification of some function terminology

- function call
- Parameter / argument
- What a function returns
 - How to “capture/store” what a function returns

function terminology

- **None** – another type in Python which is used when no value is needed.

function terminology

- **Parameter** – functions can take have 0 or more parameters. They are a way to pass data INTO a function, when it is called. The data we pass into the function is called **arguments**.
- **Calling a function** – invoking a function by giving its name in a statement in your program:
 - e.g.
 - `print("Hey")` # function call for print
 - `height = input('enter your height in feet, please')`
 - `height = int(height)`
- **Note:** A str is being passed in as an argument to print, input and int.

function terminology

- **What a function returns** – This is what the result of a method call gives.

- e.g.

```
print("Hey")           # None is returned
```

```
height = input('enter your height in feet, please')  
# str is returned
```

```
height = int(height)   # an int is returned
```

function terminology

- Let's take a look at that projected homeruns and look at each line and use the function terminology to describe them

Nested ifs

- If statements can have other if statements nested within them.
- Pay attention to the indentation to tell which if the elif or else connect to.
- Let's look at a couple of examples.

Logical operators

- **and or not** are three operators that use bools as operands and result in a bool
- Truth table for **and**

False **and** False results in False

False **and** True results in False

True **and** False results in False

True **and** True results in True

Logical operators

- **and or not** are three operators that use bools as operands and result in a bool
- Truth table for **or**

False **or** False results in False

False **or** True results in True

True **or** False results in True

True **or** True results in True

Logical operators

- **and or not** are three operators that use bools as operands and result in a bool
- Truth table for **not**

not True results in False

not False results in True