Programming: Variables and Expressions

CPSC 217: Introduction to Computer Science for Multidisciplinary Studies I
July 2024

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

June 4, 2024

UNIVERSITY OF CALGARY

Variables

- Computers manipulate data of different types:
  - Integers, real numbers, strings, and many more
- ***Variables*** **are used to store data in a particular format determined by the type**
- **A piece of memory is allocated to a variable.**
- The actual information and size of the information is **determined by the type.**
- Variables and the information stored in it will be destroyed when the program terminates.

Variables

- Some standard primitive variable types:
  - **Integer** (e.g., a = 100)
  - **Floating point** (e.g., b = 3.14)
  - **Strings** (e.g., str = ‘Hello World!’)
- **Variable initialization:**
  - ***<name of variable> = <data to be stored in the variable>***
- In Python**, the format of the data also determines the type** of the variable.
- **Function “type” is used to find a variable type**
  - Try: print(type("Hello, World!"))

UNIVERSITY OF CALGARY

Values

- Use function type to find out what class a value falls into:

```
print(type("Hello, World!"))
print(type(17))
print("Hello, World")
```

```
<class 'str'>
<class 'int'>
Hello, World
```

Using Variables

Assignment

- **a = 100**
- **b = 3.14**
- **str = “Hello World!”**

Use them in expressions

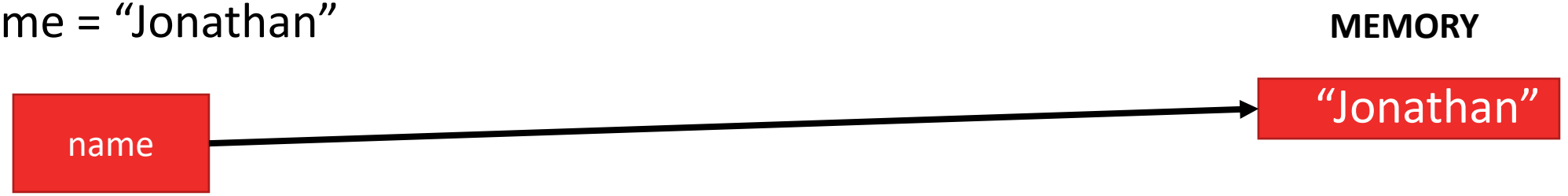
- **a = a * a** → a = 100 * 100 = 10000
- **a = a * b** → a = 10000 * 3.14 = 31400

Printing

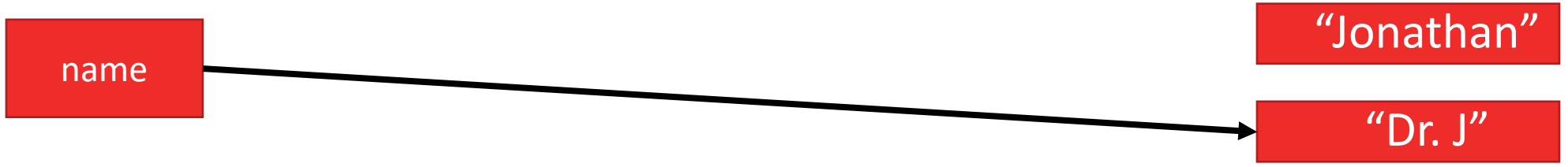
- **print (a)**
- **print (b)**

Variable Names point to data in Memory

name = “Jonathan”



name = “Dr. J”



- Changing a variable doesn’t change memory. It actually just points the name to a new memory spot with the new information. (Python eventually will get around to throwing away “Jonathan” via something called *garbage collection*)

Naming

Variable Naming

- **Python naming convention**:
  - Names **must start with a letter** (e.g., a, ..., z, A, ..., Z) (can also start with _)
  - A name **may contain any letter, any number (0, ..., 9), and the special character "_" (underscore).**
  - **White spaces and signs** with special meanings (e.g., "+", "-", "*", "/") **are not allowed.**
- **Case sensitive** "sum" is different than "Sum"
- **Cannot use reserved keywords**.
- Legal variable names: FooBar, X15Y, this_is_a_variable
- Illegal variable names: **4**2Bars, How Much, Foo**-**Bar, Var**$**
- **ALL CAPITALS is used to indicate a constant where value won't change (not enforced)**

Reserved keywords

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or
pass	raise	return	try	while	with
yield	True	False	None		

- **Cannot be used as variable name**
- This list can change as Python updates.
- We will learn the use of most of these keywords in this course.

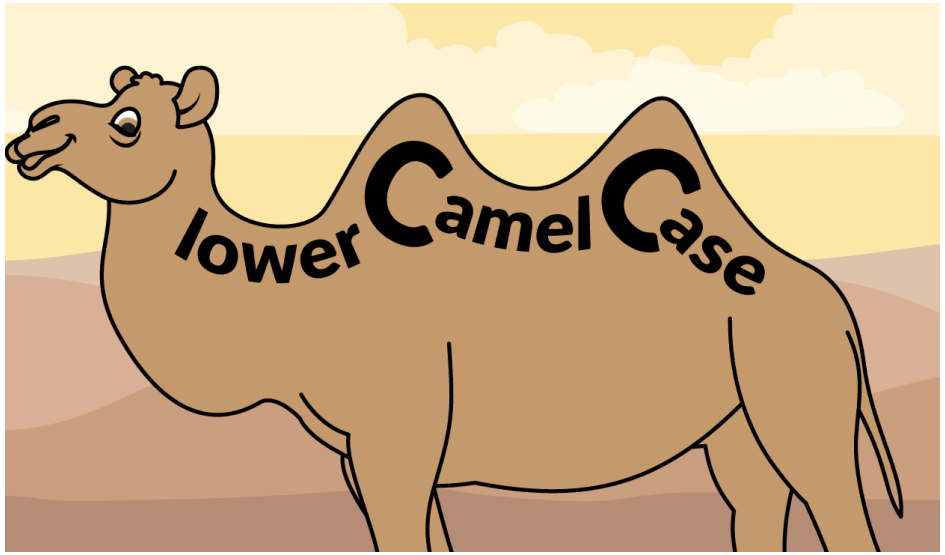
Variable Naming

- **Style requirement** *(not syntactically required)*:
  - Use meaningful names for readability.
  - Balance between brevity and descriptiveness

i	(what is this)
index	(slightly better)
indexToDatabaseOfMyCourses	(maybe too long)
indexToCourseDB	(maybe best)

- Avoid using variables that are only distinguishable by upper/lower case
  - (e.g., don't use "Sum" and "sum" in the same program)

Variable Naming (common styles)

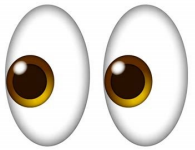


sumNetProfit

- Popular in Java, used in Python

sum_net_profit

- used more often in Python



UNIVERSITY OF CALGARY

Variable Naming kebab-case?

Meaningful names

- Use **Meaningful** names for anything that you create when programming
- If a name is meaningful to human it is not necessarily meaningful to computer

For computer there is no difference between

Average = 10 and **Ave = 10** and **Jonathan = 10**

They all require same space in memory.

Guess what this code is about!

M = 100

G = 0.2

P = M*G

Which one do you prefer?

```
m = 100
g =  0.2
p = m*g
```

```
miles = 100
gallons_of_gas_per_mile = 0.2
gallons_gas = miles * gallons_of_gas_per_mile
```

➔ Always use meaningful names!

Constants and Magic Numbers

Constants

- ***Constants*** → Variables whose **values *should not*** be changed
- Can be any type (e.g., Integers, real numbers, strings, ...)
  - In other languages: Java and C/C++, this is enforced
  - In Python, the programmer is expected to never change the value of a constant

- Constants are names using **upper case**

e.g., **PI = 3.1415**

Constant changing value

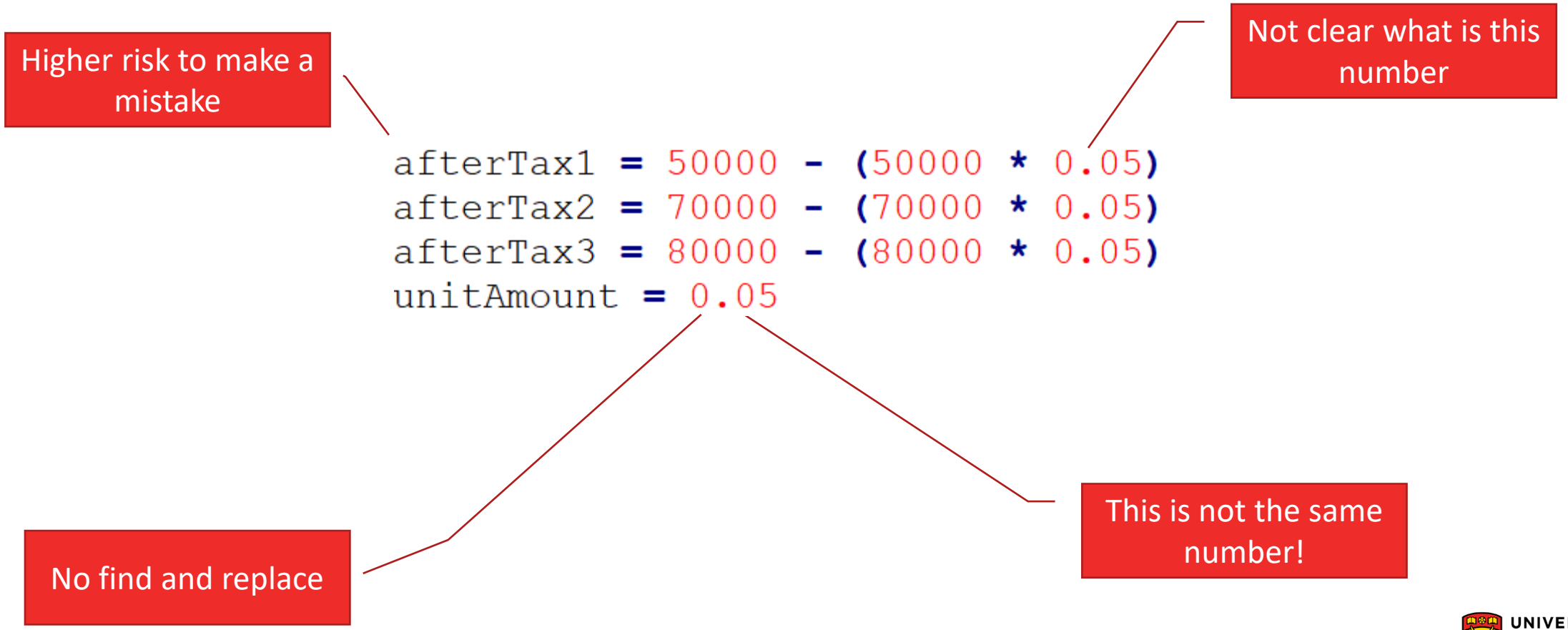
- Constant in Java (similar in C or C++)
  - **final int MY_CONSTANT = 100**
  - **MY_CONSTANT = 12**

- Constant is Python
  - **MY_CONSTANT = 100**
  - **MY_CONSTANT = 12**

Do NOT change a constant value once you define it!

Magic numbers

If I need to change the tax rate to 0.07 percent:



Using constant instead of magic numbers

Low risk to make a mistake

```
TAX_RATE = 0.05
SALARY1 = 50000
SALARY2 = 70000
SALARY3 = 80000

afterTax1 = SALARY1 - (SALARY1 * TAX_RATE)
afterTax2 = SALARY2 - (SALARY2 * TAX_RATE)
afterTax3 = SALARY3 - (SALARY3 * TAX_RATE)
unitAmount = 0.05
…
```

Clear definition of numbers

No mistake when rate is changed

No need to search and replace

Why use constants

Program becomes easier to understand and maintain

All references to the constant can be change by one single modification in constant initialization

Using constants is another way for being intelligently lazy!

Expressions

Expression

- An **expression** is a combination of values, variables, operators, and calls to functions.
- **Composed of operators and operands**
- Evaluation produce a result

- **1+1** → 2
- **len(“hello”)** → 5
- **Expressions can appear on the right-hand side of assignment statements.**
- **A value or a variable all by itself is a simple expression.**

Operators and Operands

- Operators are special tokens that represent computations like addition, multiplication and division. The values the operator works on are called operands.
- Operates:
  - **+, -, *, /** → are clear
  - ****** → exponentiation
  - **//** → integer division
  - **%** → remainder

Precedence

Precedence

Order	Operations	Precedence
1	()	Highest
2	x ** y	
3	-x, +x	
4	x * y, x / y, x % y, x // y	
5	x + y, x - y	
6	=	Lowest

The order of evaluation is determined by operator precedence (highest -> lowest)

Highest means the expression is collapse on execution of this operator first

Note: Negation to immediate to right of exponentiation has precedence

*2 ** -1 is ½ = 0.5*

Note

What is the output of the following expression?

16 - 2 * 5 // 3 + 1

Note

What is the output of the following expression?

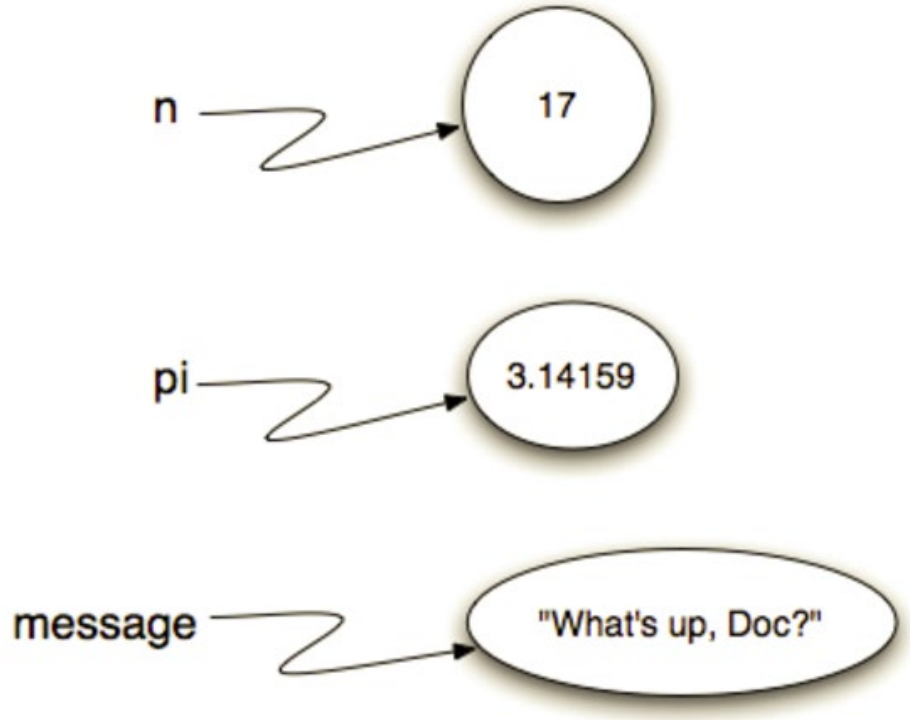
16 - 2 * 5 // 3 + 1

(16 – ((2 * 5) // 3)) + 1 → 16 – 3 + 1 = 14 ← Correct

16 – (((2 * 5) // 3) + 1) → 16 –(3 +1) = 12 ← Incorrect

Values, Variables, Statements and Expressions

- Value **type**: string, integer, float,…
- **Variables**: for keeping values
  - Naming Convention
- **Statements**: while, for, if, import,…
  - result doesn't change value in a program
- **Expressions**: combination of values and variables using operators and calls to functions
  - Produce results



Onward to … writing a program!

Jonathan Hudson
jwhudson@ucalgary.ca
https://cspages.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY