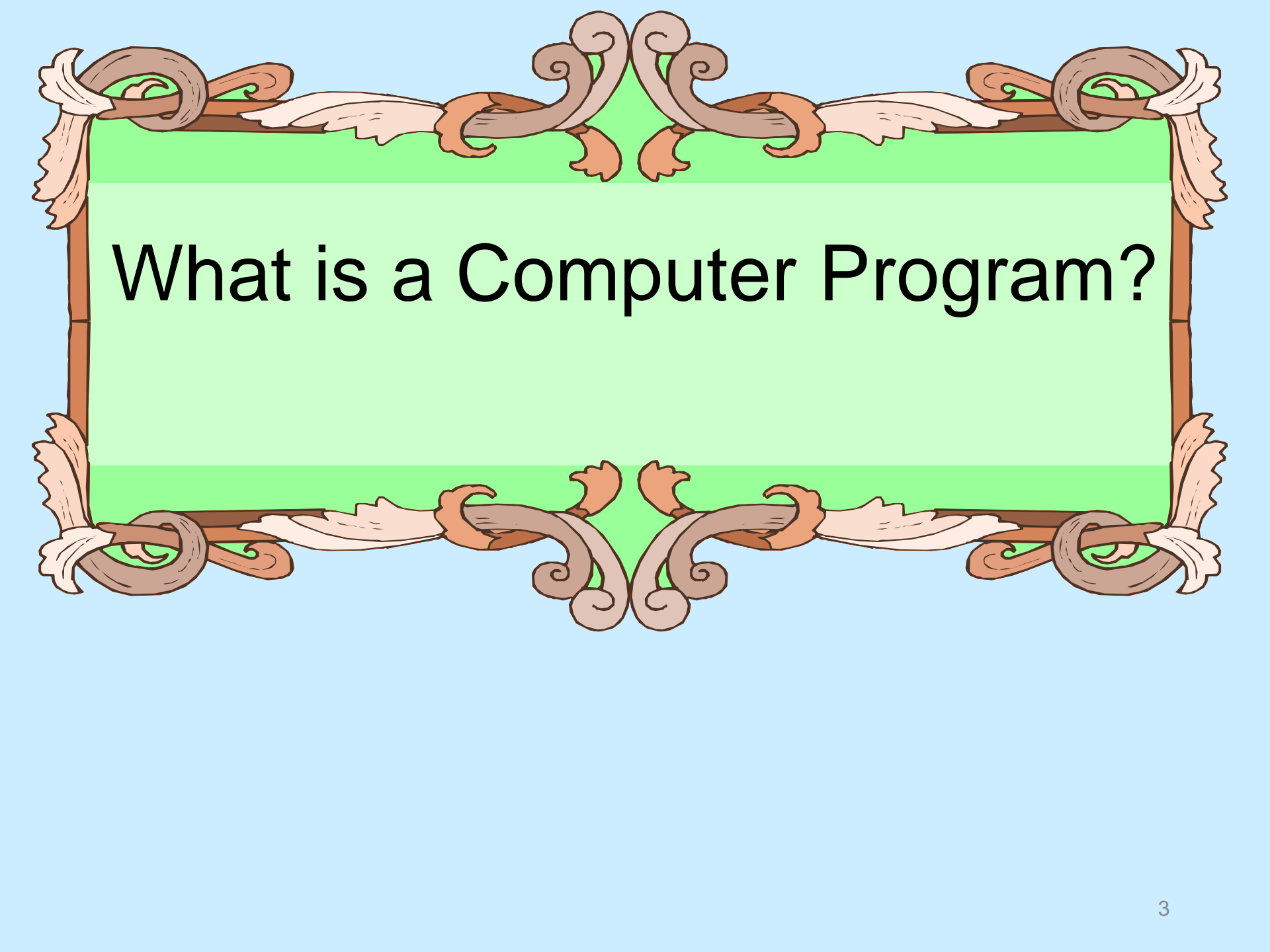


INT3075 Programming and Problem Solving for Mathematics

Python Basics

Important rules

- **Rule 1:** Think before you program
- **Rule 2:** A program is a human-readable essay on problem solving that also happens to execute on a computer
- **Rule 3:** The best way to improve your programming and problem-solving skills is to practice.



What is a Computer Program?

Program

- A program is a sequence of instructions.
- To *run* a program is to:
 - create the sequence of instructions according to your design and the language rules
 - turn that program into the binary commands the processor understands
 - give the binary code to the Operating System (OS), so OS can give the code to the processor (CPU)
 - OS tells the processor to run the program
 - when finished (or it dies :-), OS cleans up.

Interpreted

- **Python** is an *interpreted* language
- “Interpreted” means that Python looks at each instruction, one at a time, and turns that instruction into something that can be run by the computer.
- That means that you can simply open the Python interpreter and enter instructions one-at-a-time.
- You can also *import* a program which causes the instructions in the program to be executed, as if you had typed them in.

Software to be installed

- **Python 3.x**

<https://www.python.org/downloads/>


- **PyCharm (*Free Community Edition*) – An Integrated design environment (IDE)**

<http://www.jetbrains.com/pycharm/download/>

Free cloud service

- **Google colab** is a free Jupyter notebook environment that runs entirely in the cloud. It does not require a setup

<https://colab.research.google.com>

A stylized illustration of a computer monitor with a teal frame and a yellow screen. The screen displays the text "Your First Program" and "L1-1.py" in black. The monitor is on a stand and has a reflection on the surface below it.

Your First Program
L1-1.py

L1-1.py

```
1 # Calculate the area and circumference of a circle from its radius.
2 # Step 1: Prompt for a radius.
3 # Step 2: Apply the area formula.
4 # Step 3: Print out the results.
5
6 import math
7
8 radius_str = input("Enter the radius of your circle: ")
9 radius_int = int(radius_str)
10
11 circumference = 2 * math.pi * radius_int
12 area = math.pi * (radius_int ** 2)
13
14 print ("The circumference is:",circumference, \
15        ", and the area is:",area)
```

import of math

- One thing we did was to import the math module with `import math`
- This brought in python statements to support math operations
- We precede all operations of math with `math.xxx`
- `math.pi`, for example, is pi.
`math.pow(x, y)` raises x to the y^{th} power.

Getting input

The function:

```
input("Give me a value")
```

- prints “Give me a value” on the python screen and waits till the user types something (anything), ending with Enter
- **Warning:** it returns a string (sequence of characters), no matter what is given, even a number ('1' is not the same as 1, different types)

Assignment

The = sign is the assignment statement

- The value on the right is associated with the variable name on the left
- It does ***not*** stand for equality!

Conversion

Convert from string to integer

- Python requires that you must convert a sequence of characters to an integer
- Once converted, we can do math on the integers

Printing output

```
myVar = 12
```

```
print('My var has a value of: ', myVar)
```

- `print` takes a list of elements in parentheses separated by commas
 - if the element is a string, prints it as is
 - if the element is a variable, prints the value associated with the variable
 - after printing, moves on to a new line of output

At the core of any language

- Control the flow of the program
- Construct and access data elements
- Operate on data elements
- Construct functions
- Construct classes
- Libraries and built-in classes

Save as a “module”

- When you save a file, such as our first program, and place a `.py` suffix on it, it becomes a python module
- You run the module from the IDE menu to see the results of the operation
- A module is just a file of python commands

Errors

- If there are interpreter errors, that is Python cannot run your code because the code is somehow malformed, you get an error
- You can then modify the program and run it again until there are no errors

Whitespace

- For the most part, you can place white space (spaces) anywhere in your program
- use it to make a program more readable

```
1 +  
  2  
- 4
```

continuation

However, python is sensitive to end of line stuff. To make a line continue, use the \

```
print("this is a test", \
      " of continuation")
```

prints

```
this is a test of continuation
```

also, tabbing is special

- The use of tabs is also something that Python is sensitive to.
- We'll see more of that when we get to control, but be aware that the tab character has meaning to Python

Python comments

- A comment begins with a # (pound sign)
- This means that from the # to the end of that line, nothing will be interpreted by Python.
- You can write information that will help the reader with the code

Code as essay

- What is the primary goal of writing code:
 - to get it to do something
 - an “essay” on my problem-solving thoughts
- Code is something to be read by the computer. You provide comments to help readability for human.

Python keywords

Keywords:

You cannot use (are prevented from using) them in a variable name

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

Python Operators

Reserved operators in Python

+	-	*	**	/	//	%
<<	>>	&		^	~	
<	>	<=	>=	==	!=	<>

Python Punctuators

Python punctuation/delimiters

' “ # \
() [] { } @
, : . ` = ;
+= -= *= /= //= %=
&= |= ^= >>= <<= **=

Literals

Literal is a programming notation for a ***fixed value***.

- For example, 123 is a fixed value, an integer

Python name conventions

- must begin with a letter or underscore `_`
 - `Ab_123` is OK, but `123_ABC` is not.
- may contain letters, digits, and underscores
 - `this_is_an_identifier_123`
- may be of any length
- upper and lower case letters are different
 - `Length_Of_Rope` is not `length_of_rope`

Naming conventions

- The standard way for most things named in python is **lower with underline**, lower case with separate words joined by an underline:
 - this_is_a_var
 - my_list
 - square_root_function

Rule 4

We name things using conventions, but admit that, under the right circumstances, we do what is necessary to help readability.

Variable

- A variable is a name we designate to represent an object (number, data structure, function, etc.) in our program
- We use names to make our program more readable, so that the object is easily understood in the program

Variable Objects

- Python maintains a list of pairs for every variable:
 - variable's name
 - variable's value
- A variable is created when a value is assigned the first time. It associates a name and a value
- subsequent assignments update the associated value.

`my_int = 27` →

Name	Value
<code>my_int</code>	27

Namespace

- A **namespace** is the table that contains the association of a name with a value
- We will see more about namespace as we get further into Python, but it is an essential part of the language.

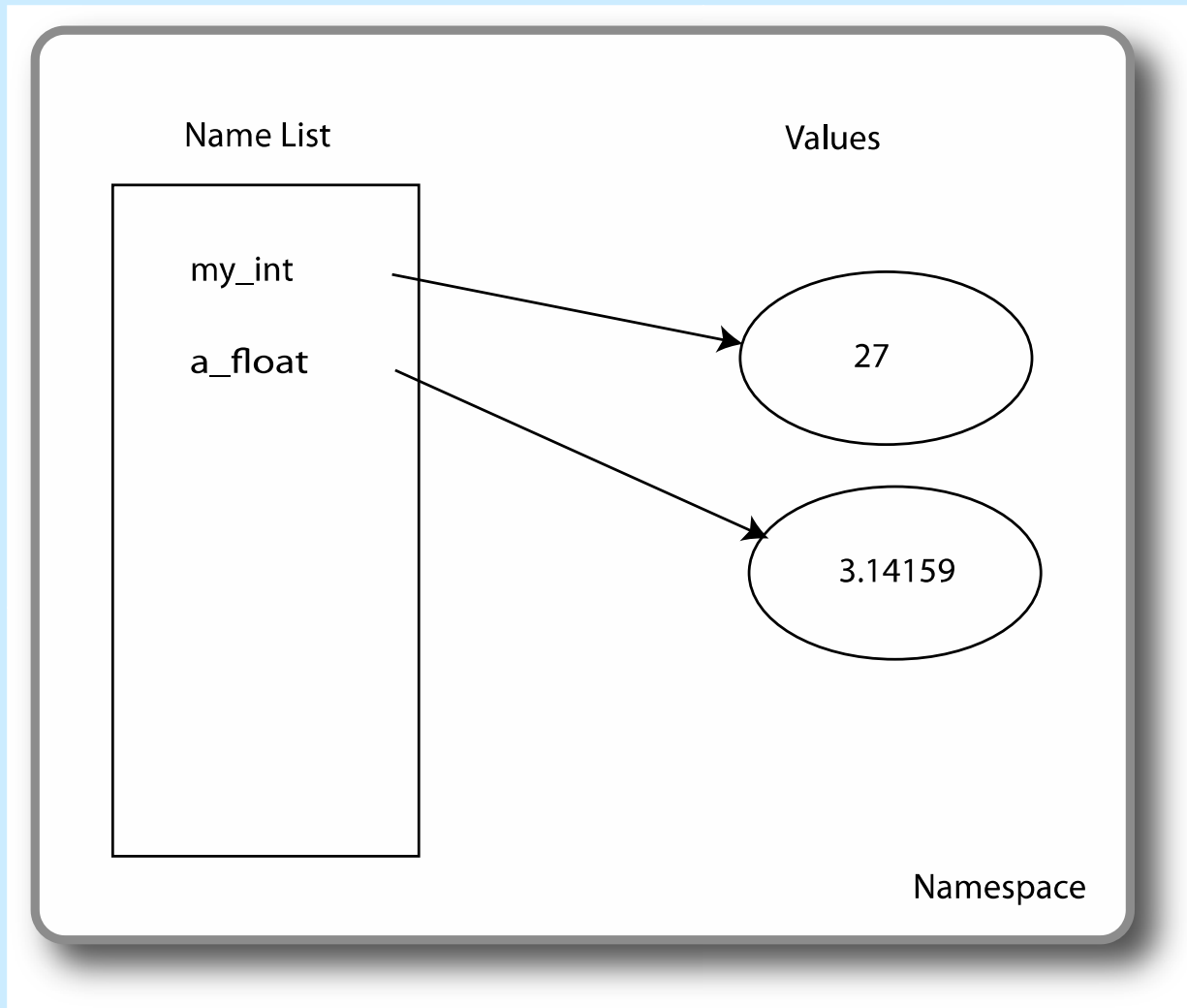


FIGURE 1.1 Namespace containing variable names and associated values.

When = doesn't mean equal

- It is most confusing at first to see the following kind of expression:

```
my_int = my_int + 7
```

- You don't have to be a math genius to figure out something is wrong there.
- What's wrong is that = doesn't mean equal

= is assignment

- In many computer languages, = means assignment.

```
my_int = my_int + 7
```

```
lhs = rhs
```

- What assignment means is:
 - evaluate the rhs
 - take the resulting value of rhs and associate it with the name on lhs

More Assignment

- **Example:** `my_var = 2 + 3 * 5`
 - evaluate expression `(2+3*5)` : 17
 - change the value of `my_var` to reference 17
- **Example** (`my_int` initially has value 2):
`my_int = my_int + 3`
 - evaluate expression `(my_int + 3)` : 5
 - change the value of `my_int` to reference 5

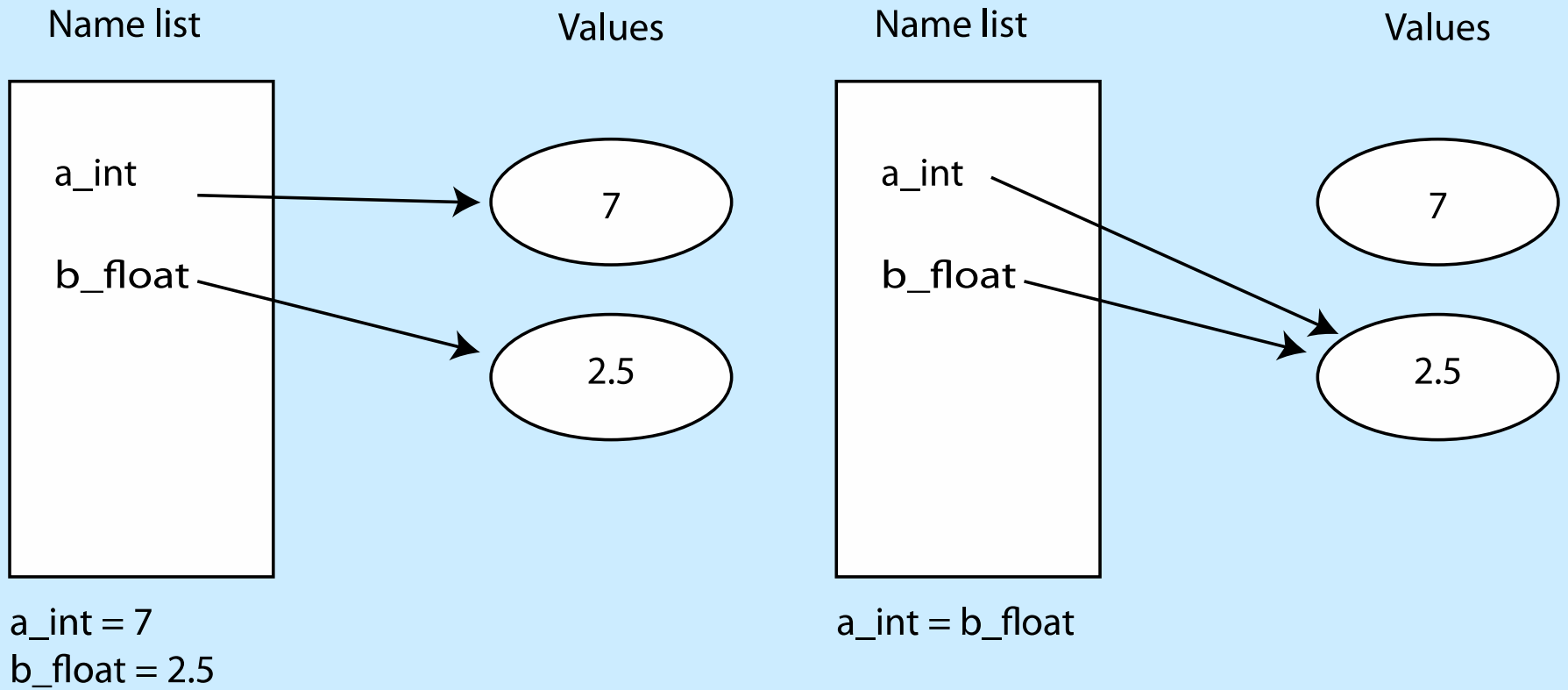


FIGURE 1.2 Namespace before and after the final assignment.

variables and types

- Python does not require you to pre-define what type can be associated with a variable
- What type a variable holds can change
- Nonetheless, knowing the type can be important for using the correct operation on a variable. Thus proper naming is important!

What can go on the lhs

- There are limits therefore as to what can go on the lhs of an assignment statement.
- The lhs must indicate a name with which a value can be associated
- must follow the naming rules

`myInt = 5`

Yes

`myInt + 5 = 7`

No

Python “types”

- integers: **5**
- floats: **1.2**
- booleans: **True**
- strings: **"anything"** or **'something'**
- lists: **[,]** **['a',1,1.3]**
- others we will see

What is a type

- a type in Python essentially defines two things:
 - the internal structure of the type (what is contained)
 - the kinds of operations you can perform
- `'abc'.capitalize()` is a method you can call on strings, but not integers
- some types have multiple elements (collections), we'll see those later

Fundamental Types

- Integers
 - `1, -27` (to `+/- 232 - 1`)
- Floating Point (Real)
 - `3.14, 10., .001, 3.14e-10, 0e0`
- Booleans (True or False values)
 - `True, False` (note the capital)

Converting types

- A character '1' is not an integer 1.
- You need to convert the string returned by the `input` command (characters) into an integer
- `int("123")` yields the integer 123

Type conversion

- `int(some_var)` returns an integer
- `float(some_var)` returns a float
- `str(some_var)` returns a string
- should check out what works:
 - `int(2.1) → 2`, `int('2') → 2`, but `int('2.1')` fails
 - `float(2) → 2.0`, `float('2.0') → 2.0`, `float('2') → 2.0`, `float(2.0) → 2.0`
 - `str(2) → '2'`, `str(2.0) → '2.0'`, `str('a') → 'a'`

Operators

- Integer

- addition and subtraction: $+$, $-$

- multiplication: $*$

- division

- quotient: $/$

- integer quotient: $//$

- remainder: $\%$

- Floating point

- add, subtract, multiply, divide: $+$, $-$, $*$, $/$

Binary operators

The operators addition(+), subtraction(-) and multiplication(*) work normally:

- `a_int = 4`

- `b_int = 2`

- `a_int + b_int` → yields 6

- `a_int - b_int` → yields 2

- `a_int * b_int` → yields 8

Two types of division

The standard division operator (/) yields a floating point result no matter the type of its operands:

- $2 / 3$ → yields 0.66666666666666666666
- $4.0 / 2$ → yields 2.0

Integer division (//) yields only the integer part of the divide (its type depends on its operands):

- $2 // 3$ → yields 0
- $4.0 // 2$ → yields 2.0

Modulus Operator

The modulus operator (%) give the integer remainder of division:

- $5 \% 3 \rightarrow 2$
- $7.0 \% 3 \rightarrow 1.0$

Again, the type of the result depends on the type of the operands.

Mixed Types

What is the difference between 42 and 42.0 ?

- their types: the first is an integer, the second is a float

What happens when you mix types:

- if you've done so, no information is lost

42 * 3 → 126

42.0 * 3 → 126.0

Order of operations and parentheses

Operator	Description
()	Parenthesis (grouping)
**	Exponentiation
+x, -x	Positive, Negative
*, /, %, //	Multiplication, Division, Remainder, Quotient
+, -	Addition, Subtraction

- Precedence of $*, /$ over $+, -$ is the same
- Remember, parentheses always takes precedence

Augmented assignment

Shortcuts may be distracting, but one that is often used is augmented assignment:

- combines an operation and reassignment to the same variable
- useful for increment/decrement

Shortcut	Equivalence
<code>my_int += 2</code>	<code>my_int = my_int + 2</code>
<code>my_int -= 2</code>	<code>my_int = my_int - 2</code>
<code>my_int /= 2</code>	<code>my_int = my_int / 2</code>
<code>my_int *= 2</code>	<code>my_int = my_int * 2</code>

Modules

Modules are files that can be imported into your Python program.

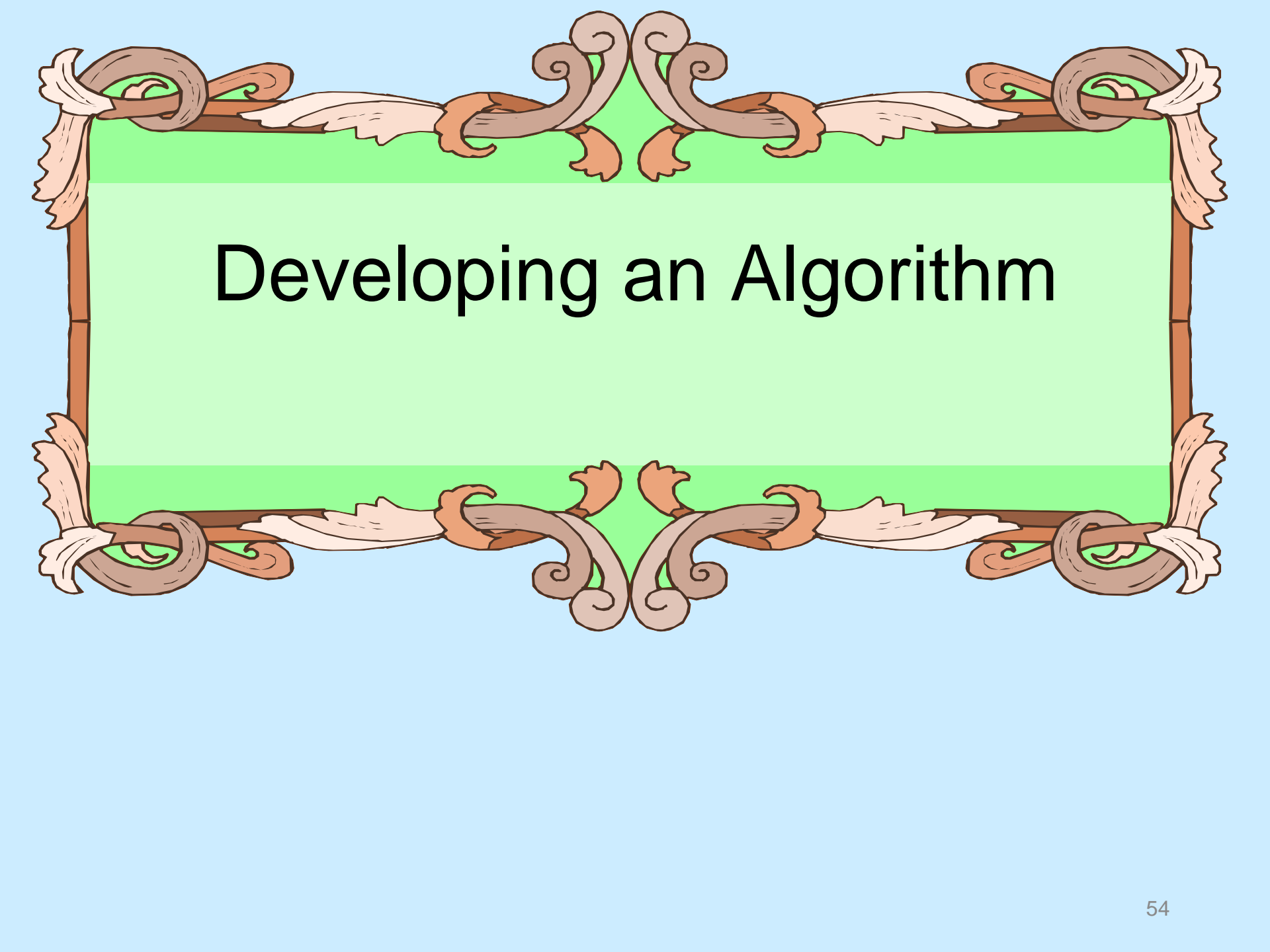
- use other, well-proven code with yours

Example is the math module

- we `import` a module to use its content
- we use the name of the module as part of the content we imported

math module

```
import math
print(math.pi)          # constant in math module
print(math.sin(1.0))    # a function in math
help(math.pow)         # help info on pow
```



Developing an Algorithm

Develop an Algorithm

How do we solve the following?

- If one inch of rain falls on an acre of land, how many gallons of water have accumulated on that acre?

Algorithm

A method – a sequence of steps – that describes how to solve a problem of class of problems

1. Find the volume (in cubic ft) of water
(where volume = depth * area)

2. Convert the volume to gallons

From Internet:

1 acre = 43,560 square feet

1 cubic foot = 7.48051945 gallons

1. Find volume in cf of water of 1 inch over 1 acre

1 inch = 1/12 foot

volume = depth * area = (1/12 * 43560 cf)

2. Convert volume in cf to gallons

gallons = volume * 7.48051945

Code Listing

L1-2.py and L1-3.py

Calculate rainfall

Calculate rainfall in gallons for some number of inches on 1 acre.

```
inches_str = input("How many inches of rain have fallen: ")
inches_int = int(inches_str)
volume = (inches_int/12)*43560
gallons = volume * 7.48051945
print(inches_int," in. rain on 1 acre is", gallons, "gallons")
```

Calculate rainfall in gallons for some number of inches on 1 acre.

```
inches_str = input("How many inches of rain have fallen: ")
inches_float = float(inches_str)
volume = (inches_float/12)*43560
gallons = volume * 7.48051945
print(inches_float," in. rain on 1 acre is", gallons, "gallons")
```

Rule 5

Test your code, often and thoroughly!

One thing we learn in writing our code is that we must test it, especially against a number of conditions, to assure ourselves that it works

The Rules

1. Think before you program
2. A program is a human-readable essay on problem solving that also happens to execute on a computer.
3. The best way to improve your programming and problem-solving skills is to practice.
4. Do what is necessary to help readability
5. Test your code, often and thoroughly!