Python Programming

Python

Running a Python Program

- **Interactive Mode**: Python commands are typed directly into the *Python shell*, where they get immediately executed. Useful for quick experimenting

- **Normal Mode**: Alternatively, Python commands can be saved in a file whose name ends with **.py**, and then executed as often as you like

Python

Defining Your Own Functions

Printing in a Terminal Console

- Examples of console output statements using the simplest Python function, *print*:
  - **print (' ')**
  - **print (' ', end='')**

- Useful escape sequences: \n \t \"

Define Your Own Python Function

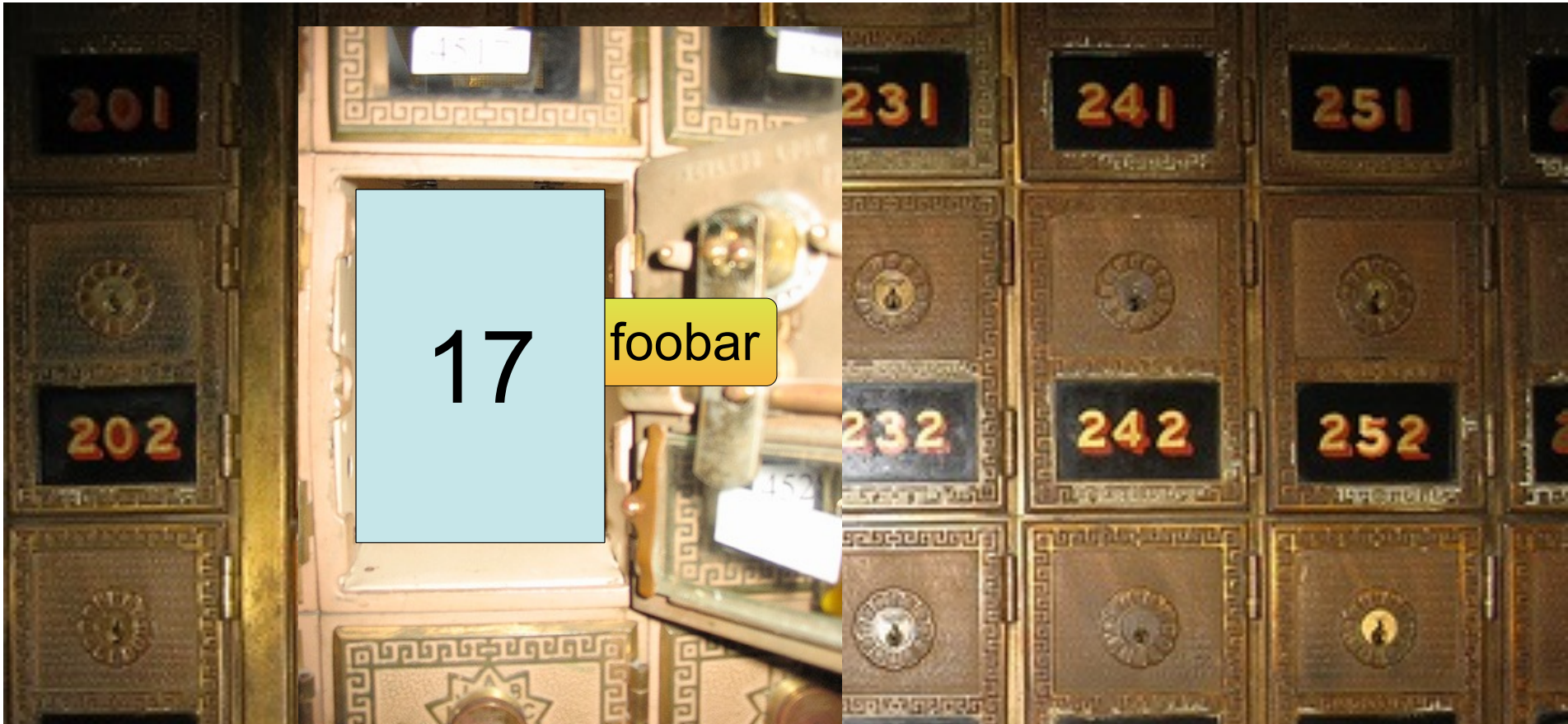
- Draw a box that looks like this using *print*:

Now modify the program to draw more than one of these rectangles.

Python

Variables

Computer Memory and Variables



Variables Get Values Using =

- The first time a variable is *assigned* a value, the variable is created and initialized with that value. After a variable has been defined, it can be used in other statements. E.g.,
  - `foobar = 17`
  - `foobar = 3.14`
  - `foobar = 'blah'`

- Note: The data type of a value specifies how the value is stored in the computer and what operations can be performed on the value.

Naming Identifiers in Python

- Identifiers must begin with a letter and may contain additional letters and digits. __ (underscore) can be used in place of a letter. Are any of following legal identifiers?
  - `_6pack`
  - `x+y`
  - `president Biden`
- Avoid **l** (lowercase letter el), **O** (uppercase letter oh), or **I** (uppercase letter eye) as single character variable names.
- Reserved words (e.g., *for*, *while*, *def*, …) cannot be used!
- Function and variable names should be in lowercase.

The Assignment Statement

- You *assign* the value of an expression to a variable: *variable* **=** *expression*
- *Assignment* is different from algebraic "equals"

Algebra	Python
x = 3	x = 3
y = 2x	y = 2 * x
x = 5	x = 5
$0 = x^2 - x - 2$	

Arithmetic Operators

- Simple arithmetic expressions can be formed with

- With //, the answer is truncated. For positive integers, floor division computes the quotient and discards the fractional part.
- % (modulus) computes the remainder
- Suppose n = 1729 What do the following expressions compute?
  - **n % 10** **n // 10**
  - **n % 100** **n % 2**
- // and % are also defined for negative integers and floating point numbers, but won't be covered!

Python

Iteration / Looping / Repetition

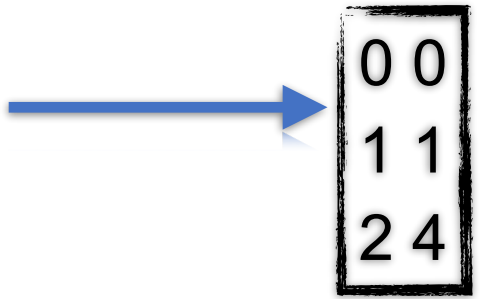
Iteration Using the **for** Loop

- Python syntax

```
for variable in container :
    Python statement[s]
        # statements in the loop body are
        # executed for each element in the container
```

- An example:

```
for i in range(3):
    print(i, i*i)
```



Another *for* Loop Example

```
print ("+----+")
for i in range(3):
    print ("\\    /")
    print ("/    \\")
print ("+----+")
```

outputs

```
+----+
\    /
/    \
\    /
/    \
\    /
/    \
+----+
```

The *range* Function

- **range** function accepts 1, 2, or 3 arguments
-
```
for x in range(5):
    print (x)
```

 # outputs 0, 1, 2, 3, 4

-
```
for x in range(1, 5):
    print (x)
```

 # outputs 1, 2, 3, 4

-
```
for x in range(4, 14, 3):
    print (x)
```

 # outputs 4, 7, 10, 13

Counting Backwards

How to produce the lyrics:

99 bottles of beer on the wall.

99 bottles of beer!

If one of those bottles should happen to fall,

There'd be 98 bottles of beer on the wall!

98 bottles of beer on the wall.

98 bottles of beer!

If one of those bottles should happen to fall,

…

Python

Keyboard Input

Getting Keyboard Input

- Your programs will be more flexible if they ask the user for inputs rather than using fixed values.
- When a program requests user input, it should first print a message that tells the user what value is expected. Such a message is called a *prompt*. In Python, displaying a prompt and reading the keyboard input is combined in one operation using the **input()** function

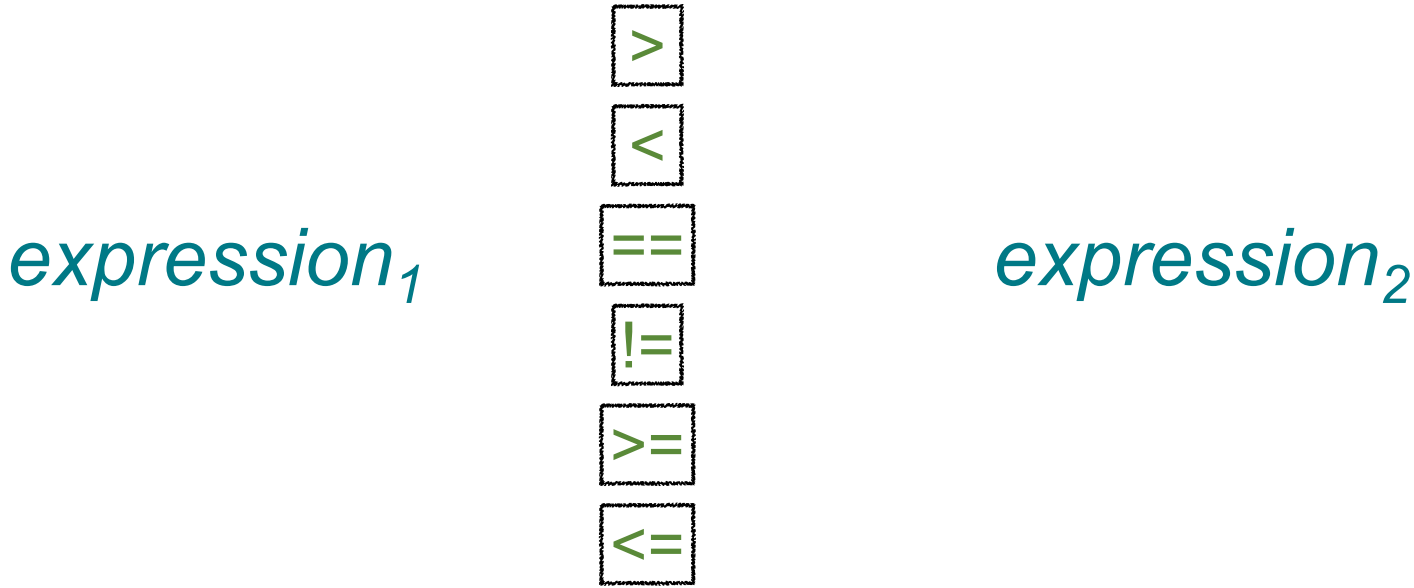
Simple Example Using input()

```
#file greeting.py

name = input ("What's your name? ")
age = input ("How old are you? ")
print ("Greetings", name)
print ("You don't look", age, "years old!")
```

Boolean-valued Expressions

- In Python, the 6 basic *relational* operators each produce the value *True* or *False*:



Conditional Statements

- Syntax

```
if Boolean-expression :
        Python-statement[s]1
```

```
if Boolean-expression :
        Python-statement[s]1

else:
        Python-statement[s]2
```

- **elif** is an abbreviation of "else if"

Python

Defining Functions that Accept Arguments

Defining Python Functions That Accept Arguments

- Functions can have multiple parameters (separated by ,)
  - When calling the function, you must pass an actual value for each parameter.
- Defining

```
def function_name ( param1, param2, ..., paramn ) :
        Python statement[s]
```

- Calling: function_name (expr1, expr2, … , exprn)

Python

Modules and Indefinite Repetition

3 Ways to Import Modules

- You can import multiple functions from the same module:
  - **from math import sqrt, sin, cos**
- You can also import the entire contents of a module:
  - **from math import ***
- Alternatively, import the module with the statement
  - **import math**
  - With this form of import, you need to add the module name and a period before each function call, like this: y = math.sqrt(x)

math Module Methods That Return Values

Function	Description	Example	Result
gcd	greatest common divisor	math.gcd(32, 72)	8
log10	logarithm base 10	math.log(1000)	3
sqrt	square root	math.sqrt(3)	1.7320508075688
sin	sine (radians)	math.sin (3*math.pi / 2)	-1.0
factorial	product of [1.. n]	math.factorial(6)	720
degrees	radians to degrees	math.degrees(3.14)	179.908747671079

Iteration Using **while**

- The while permits "indefinite" looping. The syntax:

```
while  bool-expression :
        Python instruction[s]
```

- The semantics: *bool-expression* gets evaluated. If *True*, the Python instruction[s] are executed; then we start over again by checking the bool-expression . The *Python instruction[s]* may be executed any number of times.
  - If *bool-expression* is False initially, then … ?
  - After the **while** loop terminates, *bool-expression* … ?
  - If executing *Python instructions[s]* does not make it possible for *bool-expression* to become False (when it started off True), then what?

Python

Formatted Output

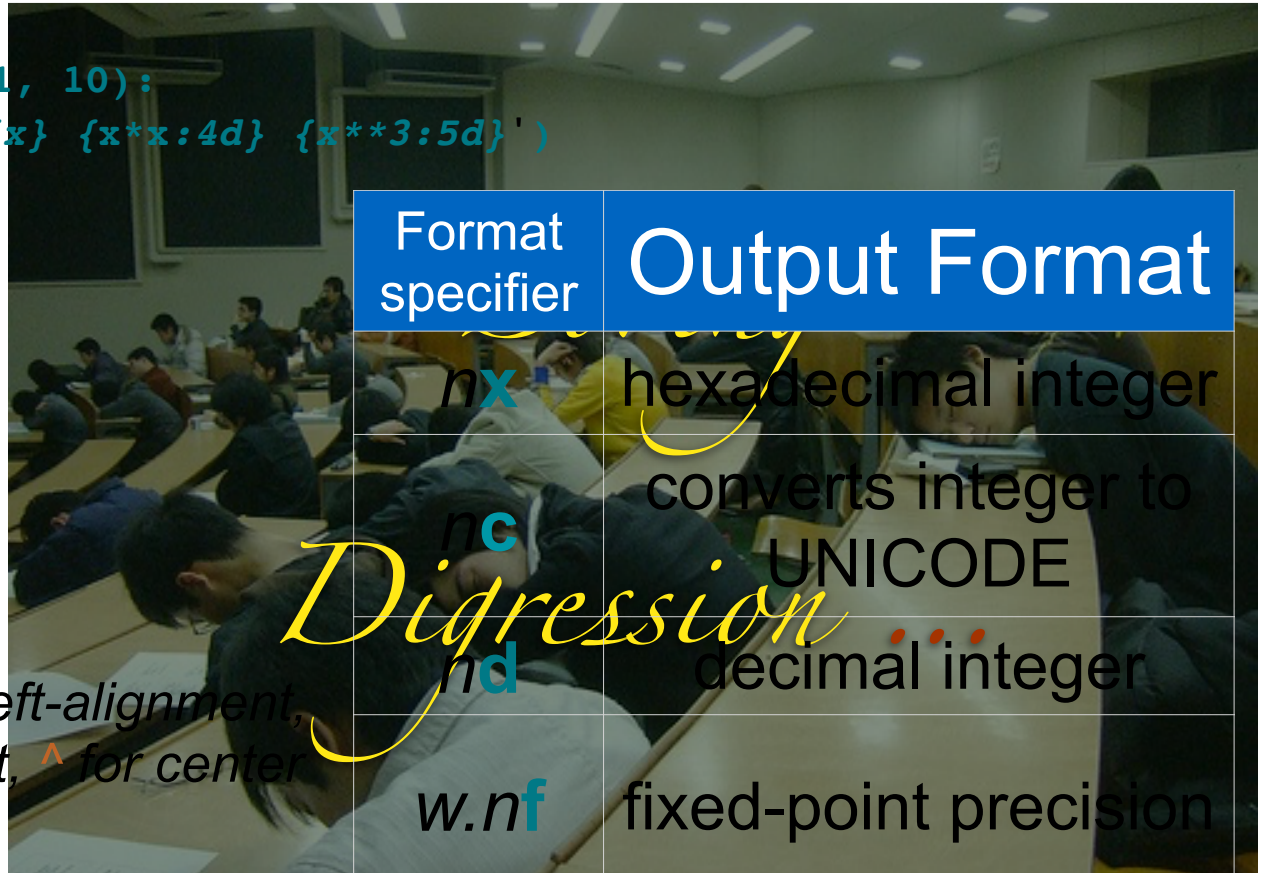
Formatted Output Via f-strings

-
```
for x in range(1, 10):
    print (f'{x} {x*x:4d} {x**3:5d}')
```

outputs

```
1    1    1
2    4    8
3    9   27
4   16   64
5   25  125
6   36  216
7   49  343
8   64  512
9   81  729
```

optional: use < for left-alignment, > for right-alignment, ^ for center alignment



Format specifier	Output Format
*n*x	hexadecimal integer
*n*c	converts integer to UNICODE
*n*d	decimal integer
*w.n*f	fixed-point precision

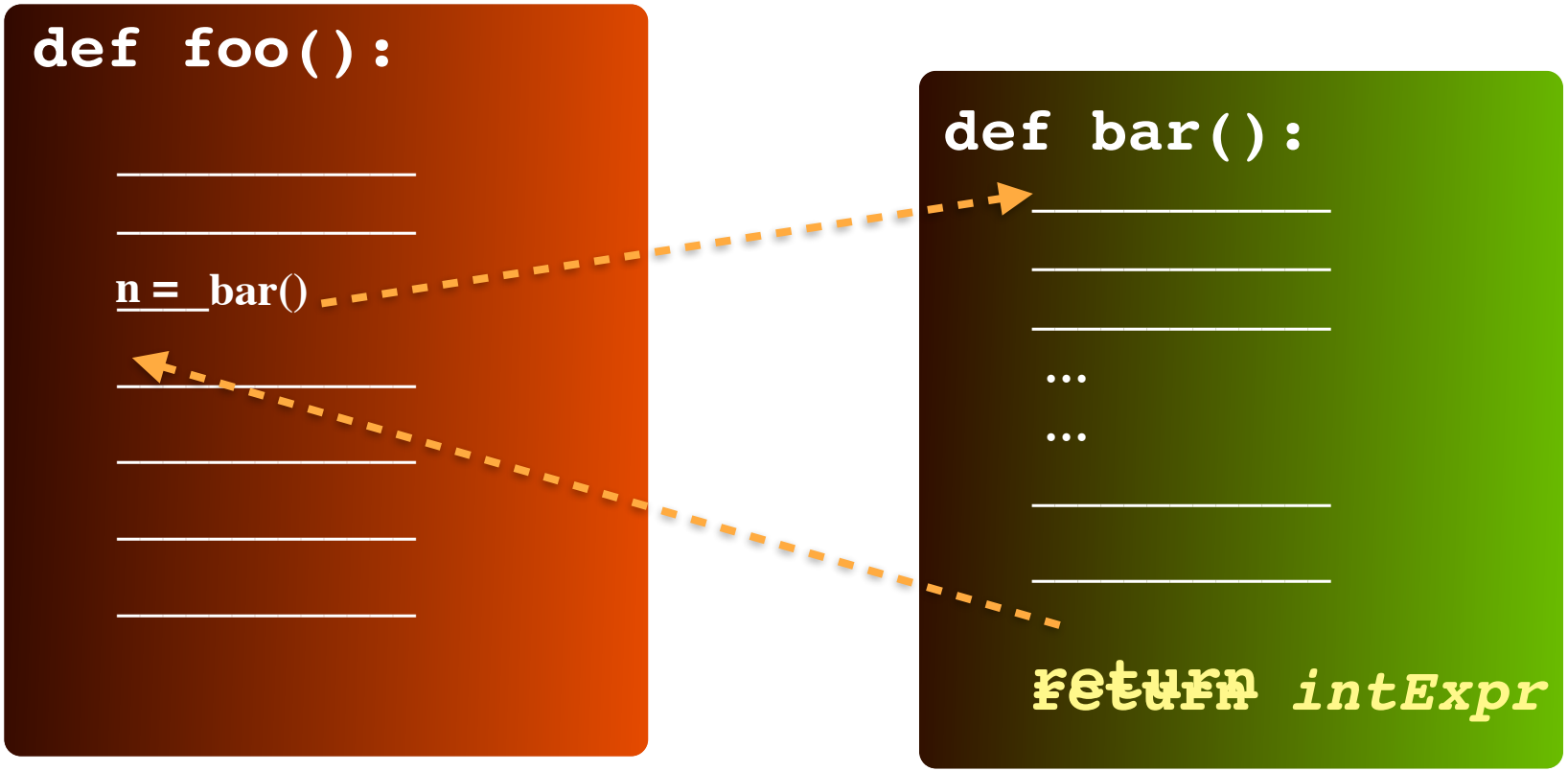
Python

Review

Overview of Basic Python Concepts

- Output to the console using **print** function
- Variables can obtain simple values (*int*, *float*, *bool*, *str*) via assignment
- Arithmetic operators (+, -, *, /, //, %, **)
- Conditions and relational operators (<, >, <=, >=, !=, ==)
- **for** loop and the **range** function
- **while** loop
- writing functions that have 0 or more parameters
- the **input** function to prompt the user for keyboard input
- import modules (**math**, **random**, ...)

How Functions "Return"



Writing Functions that Return a Value

- Use at least one **return** statement with a value
- Example: compute area of a circle
  - ```
    def circle_area (radius) :
        result = math.pi * radius*radius
        return result
                    # see file returnValues.py
    ```
  - However, this will NOT work, unless you *import math*
    ```

Python

Logical Operators

Logical Operator: *and*

- To execute code only if both of two (or more) conditions are true, put *and* between the conditions.
  - Example: does integer n satisfies the inequality $4 < n <= 9$ use `if 4 < n and n <= 9:` …
- A "truth table" formally defines and :

p	False	False	True	True
q	False	True	False	True
p and q	False	False	False	True

Logical Operator: *or*

- If at least one of two (or more) conditions need be true, use the "logical inclusive or" operator, or
  - Example: if age < 16 or age > 65 :
    print ("Not in workforce")

- A "truth table" formally defines or :

p	False	False	True	True
q	False	true	False	True
p or q	False	True	True	True

Logical Operator: *not*

- Sometimes you may want to invert a condition with **not** (logical negation)
  - This operator takes a single boolean expression and evaluates to True if that condition is False, and to False if that condition is True.
- Occasionally useful are “DeMorgan’s Laws”
  - `not ( a and b )` is the same as …
  - `not ( a or b )` is the same as …