Every Boilermaker Engineer Codes: 101 Entry-Level Programming in Python

Lecture 01

Dr. John H. Cole

`<jhcole@purdue.edu>`

PURDUE

U N I V E R S I T Y ®

COLLEGE OF ENGINEERING

Fall 2021

Part I

ENTRY-LEVEL PROGRAMMING IN PYTHON

Who - Instructor

Instructor:

- Prof. John Cole (he/him)
- jhcole@purdue.edu
- Hometown: Lafayette, Indiana
- Major: BSEE, PhD Electrical Engineering
- Personal: Dog dad to Beau and Bear

Who - Peer Teachers

Sections	Name	Email
1, 3	Mark Bogdanov	bogdanov@purdue.edu
2, 4, 5	Thirawat Bureetes (Tam)	tbureete@purdue.edu
5	Stephanie Close	closes@purdue.edu
1	Deep Gada	dgada@purdue.edu
1, 3, 4, 5	Advait Jawaji (AJ)	ajawaji@purdue.edu
1, 2, 5	Darren Lie	lied@purdue.edu
2, 3	Yue Lin (Cassie)	lin1000@purdue.edu
3, 4	Carly Mendenhall	cmenden@purdue.edu
2, 3, 4, 5	Aletea Vanveldhuisen (Tea)	avanveld@purdue.edu
4	Minjun Zhang (Jess)	zhan3624@purdue.edu

What are We Learning

Structured Programming

- variables, sequential statement, control structures, loops
- functions, modules and the Python standard library
- lists, tuples, dictionaries, and sets
- strings, file input and output
- plotting with matplotlib

Object-Oriented Programming

- classes, objects, attributes, and methods
- abstraction, encapsulation, inheritance, and polymorphism

Where

Class

- On Brightspace
  https://purdue.brightspace.com/d2l/home/401876
- Check under the "All" tab for "Fall 2021
  EBEC Entry-Level Programming in Python"

Office Hours

- Office hours will be held via Microsoft Teams
- use team code uahux0x

Q&A and Discussion Forum

- On Campuswire
  https://campuswire.com/c/G442E6951/feed/
- Please add <team@campuswiremail.com> to your email safe
  list so you won't miss any important announcements.

When

Class

- August 30th - December 12th
- New modules released every Monday (even on holidays)
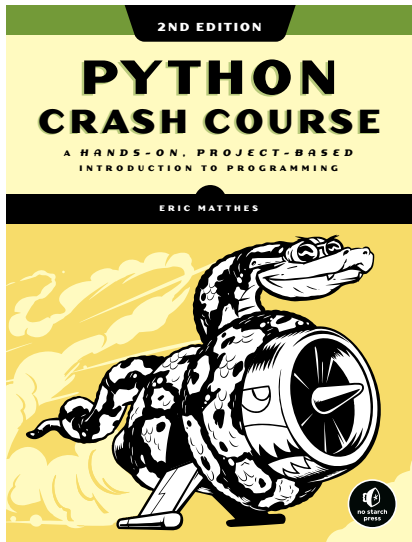- Assignments due second Monday after being released

Office Hours

- To Be Determined

Q&A and Discussion Forum

- anytime

How to Get the Textbook

Textbook:

- digital access available FREE at Purdue library
- Publisher: No Starch Press; 2nd Edition (May, 2019)
- ISBN-10: 1-59327-928-0
- ISBN-13: 9781593279288

How You Will Be Graded

50 % Programming Exercises
- 3-6 programming exercises weekly on Gradescope
- submit code and screenshots for each exercise
- 100 points per exercise (90 autograded, 10 manual)

25 % Quizzes
- weekly on Brightspace
- 10-15 multiple choice questions

25 % Final Project
- 2 weeks
- comprehensive

Notes:
- This is a FREE non-transcript course.
- This course will not affect your GPA.

How to Earn the Certificate

CERTIFICATE ISSUED BY COLLEGE OF ENGINEERING

- Score 70 % or higher to earn a certificate issued by the College of Engineering
- certificate file will be hosted long-term by Purdue for verification
- recipients will receive a link to their certificate file by email
- print your certificate or include the link on your resume, LinkedIn profile, or wherever

Thanks for watching!

Running Python
0000

Processing
0000000000000000000

Input
000

Output
00000000

Part II

PROGRAMMING IN PYTHON

OPEN A TERMINAL

- Linux & Mac

  Terminal

  ```
  user@machine:~$
  ```

- Windows (also called command prompt)

  Terminal

  ```
  C:\Users\user>
  ```

- Generic Terminal

  Terminal

  ```
  $
  ```

Running Python
○●○○

Processing
○○○○○○○○○○○○○○○○○○○○○

Input
○○○

Output
○○○○○○○○

START INTERACTIVE MODE

Terminal

```
$ python
Python 3.8.2 (default, Jul  16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license"
for more information.
>>>
```

The Python interactive prompt ">>> "

- indicates interpreter is waiting for a statement to be typed
- reappears after previous statement is executed
- good way to experiment and learn
- exit by pressing ctrl + D or entering exit()
- statements are not saved

Running Python
○○●○

Processing
○○○○○○○○○○○○○○○○○○○

Input
○○○

Output
○○○○○○○○

INTERACTIVE MODE - EXAMPLE

Terminal

```
$ python
Python 3.8.2 (default, Jul  16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license"
for more information.
>>> print('hello world')
hello world
>>> exit()
$
```

Running Python
○○○●

Processing
○○○○○○○○○○○○○○○○○○○

Input
○○○

Output
○○○○○○○○

SCRIPT MODE

- type Python statements in a file
- save the file with the .py extension
- run by typing "python filename.py" in a terminal

Editor - hello_world.py

```
1 print('Hello World!')
```

Terminal

```
$ python hello_world.py
Hello World!
$
```

Math Operators

Math Operator performs a calculation

  Operands the values an operator acts on

Common Math Operators

- addition "+"
- subtraction "-"
- multiplication "*"
- float division "/"
- exponentiation "**"
- floor division "//"
- modulus (remainder) "%"

Math Operator Examples

Terminal

```
$ python3
>>> 5 + 10
15
>>> 3 - 7
-4
>>> 6 * 15
90
>>> 7 / 4
1.75
>>> 7 // 4
1
>>> 7 % 4
3
>>>
```

Terminal

```
>>> 4 / 1.5
2.6666666666666665
>>> 4 // 1.5
2.0
>>> 2**8
256
>>> 12 / 3*2
8.0
>>> 12 / (3*2)
2.0
>>> 2 + 3*(9-4)**2
77
>>>
```

Running Python
○○○○

Processing
○○●○○○○○○○○○○○○○○○○○

Input
○○○

Output
○○○○○○○○

OPERATOR PRECEDENCE

- order of operations is determined by precedence
- higher precedence operators execute first
  1. parentheses ()
  2. exponents **
  3. multiplication *, division / and //, and modulus %
  4. addition + and subtraction -
- equal precedence operators execute from left to right

Running Python
oooo

Processing
ooo●ooooooooooooooooo

Input
ooo

Output
oooooooo

VARIABLES

VARIABLE a name that references a value stored in memory

- think of them as handles for your data
- used to access and manipulate data in memory
- can point to any type of data

ASSIGNMENT OPERATOR the equal sign "="

EXPRESSION anything that results in a value (e.g. 1+2)
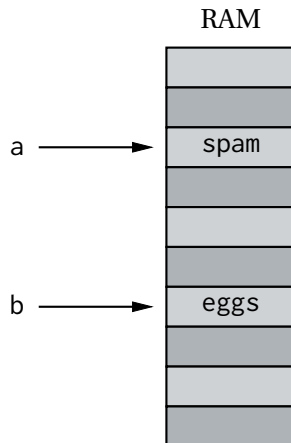
LITERAL a value written directly in a program

ASSIGNMENT STATEMENT sets the value that a variable refers to

- general form is: variable = expression
- e.g. age = 21 (variable, assignment operator, literal)

Simple Assignment

Terminal

```
$ python
>>> a = 'spam'
>>> a
'spam'
>>> b = 'eggs'
>>> b
'eggs'
>>> a = b
>>> a
'eggs'
>>>
```

RAM

a ——————→ spam

b ——————→ eggs

Simple Assignment
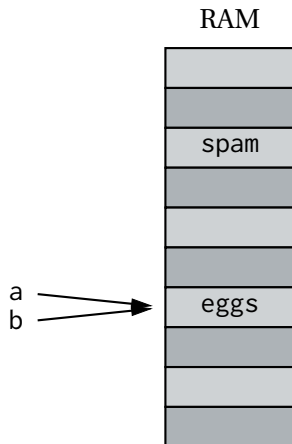
Terminal

```
$ python
>>> a = 'spam'
>>> a
'spam'
>>> b = 'eggs'
>>> b
'eggs'
>>> a = b
>>> a
'eggs'
>>>
```

RAM
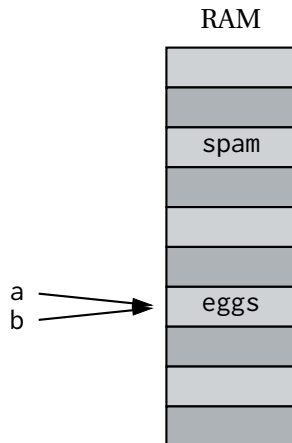
GARBAGE COLLECTION

What happens to the 'spam'
stored in memory?

- values that are no longer
  referenced by a variable are
  periodically removed
- this process is called
  garbage collection
- automatically frees up
  memory so it can be used
  for something else

RAM

Running Python
○○○○

Processing
○○○○○○●○○○○○○○○○○○

Input
○○○

Output
○○○○○○○○

VARIABLES AS ARGUMENTS

FUNCTION   a piece of pre-written code that performs an operation

ARGUMENT   data given to a function

- A variable can be passed as an argument to a function.

Terminal

```
$ python
>>> a = "spam"
>>> print(a)
spam
>>>
```

Variables Must Be Assigned

Variables can only be used after being assigned.

Terminal

```
$ python
>>> print(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
>>> a = "eggs"
>>> print(a)
eggs
>>>
```

VARIABLE TO LEFT OF ASSIGNMENT

The left side of an assignment statement must be a variable.

Terminal

```
$ python
>>> 1 = a
  File "<stdin>", line 1
SyntaxError: can't assign to literal
>>> a = 1
>>>
```

MULTIPLE ASSIGNMENT & OPERATORS WITH VARIABLES

- multiple assignment
  matches variables and
  values one-for-one

Terminal

```
>>> a, b, c = 5, 7, 9
>>> a
5
>>> b
7
>>> c
9
```

- operators work with
  variables just like they do
  with values

Terminal

```
>>> a, b, c = 5, 7, 9
>>> a + b
12
>>> c % b
2
>>> c % b * a
10
```

COMPOUND ASSIGNMENT OPERATORS

Terminal

```
>>> a = 100
>>> a = a + 10
>>> a += 10
>>> a
120
>>> a *= 2
>>> a
240
>>> a //= 5
>>> a
48
>>> a -= 6
>>> a
42
```

- a variable can appear on both sides of an assignment
- a += 3 is equivalent to a = a + 3
- a -= 3 is equivalent to a = a - 3
- a *= 3 is equivalent to a = a * 3
- a /= 3 is equivalent to a = a / 3
- a **= 3 is equivalent to a = a ** 3
- a //= 3 is equivalent to a = a // 3
- a %= 3 is equivalent to a = a % 3
- python does not have ++ or -- operators

Running Python
○○○○

Processing
○○○○○○○○○○○○●○○○○○○○

Input
○○○

Output
○○○○○○○○

ALL VALUES HAVE A TYPE

The `type()` function returns the type of its argument.

```
Terminal

>>> type(5)
<class 'int'>
>>> type(5.0)
<class 'float'>
>>> type('eggs')
<class 'str'>
>>> type("spam")
<class 'str'>
>>> type('''3''')
<class 'str'>
```

Basic Types:

INT integers (e.g. 1, 35, -10)

FLOAT floating point numbers
(e.g. 3.14159, 10.0, 1e5)

STR character strings,
anything enclosed in
single, double, or triple
quotes (e.g. "eggs",
"spam", "3")

All Variables Have A Type

Terminal

```
>>> a, b, c = 5, 7.0, 'eggs'
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(c)
<class 'str'>
>>> c = 3
>>> type(c)
<class 'int'>
```

Variables:

- have the same type as the value they refer to
- can refer to a value of any type
- can change their value and type at any time

MIXED TYPE OPERATIONS

Terminal

```
>>> 5+2
7
>>> 5+2.0
7.0
>>> 5/2
2.5
>>> 5//2
2
>>> 5%2
1
>>> 5.1%2
1.0999999999999996
>>>
```

Terminal

```
>>> type(5+2)
<class 'int'>
>>> type(5+2.0)
<class 'float'>
>>> type(5/2)
<class 'float'>
>>> type(5//2)
<class 'int'>
>>> type(5%2)
<class 'int'>
>>> type(5.1%2)
<class 'float'>
>>>
```

Running Python
○○○○

Processing
○○○○○○○○○○○○○○○●○○○○

Input
○○○

Output
○○○○○○○○

Type Conversions

Terminal

```
>>> a = 1.5
>>> type(a)
<class 'float'>
>>> b = str(a)
>>> type(b)
<class 'str'>
>>> b
'1.5'
>>> c = float(b)
>>> type(c)
<class 'float'>
>>> c
1.5
```

Terminal

```
>>> d = int(c)
>>> type(d)
<class 'int'>
>>> d
1
>>>
```

- str() returns its argument as a string
- float() returns its argument as a float
- int() returns its argument as an integer

int() LIMITATIONS

- int() can not handle strings with decimal points

```
Terminal
```

```
>>> int('5')
5
>>> int(5.8)
5
>>> int('5.8')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10:
'5.8'
>>>
```

float() LIMITATIONS

- float() can only handle integers and numeric strings

Terminal

```
>>> float(5)
5.0
>>> float('5.8')
5.8
>>> float('five')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: 'five'
>>>
```

QUOTES IN STRINGS

Terminal

```
>>> "but I don't like spam"
"but I don't like spam"
>>> 'The nights who say "Ni!"'
'The nights who say "Ni!"'
>>> 'Don't say "Ni!"'
  File "<stdin>", line 1
    'Don't say "Ni!"'
          ^
SyntaxError: invalid syntax
>>> 'Don\'t say "Ni!"'
'Don\'t say "Ni!"'
>>> "Don't say \"Ni!\""
'Don\'t say "Ni!"'
```

- single quotes must be escaped within single quotes
- double quotes must be escaped within double quotes
- use backslash \ to escape quotation marks

Running Python
○○○○

Processing
○○○○○○○○○○○○○○○○○○●

Input
○○○

Output
○○○○○○○○

MORE ABOUT STRINGS

Terminal

```
>>> '''Don't say "Ni!"'''
'Don\'t say "Ni!"'
>>> """And now
... for something completely different"""
'And now\nfor something completely different'
>>> "egg" + "s a" + "nd s" + "pam"
'eggs and spam'
```

- triple quoted strings can contain both single and double quotes
- triple quoted strings can span multiple lines
- can enter special characters newline '\n' and tab '\t'
- strings can be combined with the + operator

Running Python
○○○○

Processing
○○○○○○○○○○○○○○○○○○

Input
●○○

Output
○○○○○○○○

THE input() FUNCTION

Terminal

```
>>> name = input('What is your name? ')
What is your name? Sir Arthur
>>> name
'Sir Arthur'
>>>
```

- enables reading input from the keyboard
- general form is variable = input(prompt)
- no automatic space after prompt
- input() always returns a string

READING NUMBERS WITH THE input() FUNCTION

Use a type conversion to input a number

Terminal

```
>>> age = input('How old are you? ')
How old are you? 7
>>> age
'7'
>>> type(age)
<class 'str'>
>>> age = int(age)
>>> age
7
>>> type(age)
<class 'int'>
>>>
```

READING NUMBERS WITH THE input() FUNCTION CONT.

- can use nested function calls func_a(func_b(argument))
- invalid input will produce and error

Terminal

```
>>> age = int(input('How old are you? '))
How old are you? 7
>>> type(age)
<class 'int'>
>>> age = int(input('How old are you? '))
How old are you? 7.5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10:
'7.5'
```

Running Python
○○○○

Processing
○○○○○○○○○○○○○○○○○○

Input
○○○

Output
●○○○○○○○

THE print() FUNCTION

- can have multiple input arguments separated by commas
- arguments are displayed in the order they are passed
- by default, arguments are displayed separated by a space
- keyword argument sep='delimiter' changes the space between arguments to delimiter
- keyword argument end='delimiter' changes the newline at the end to delimiter

Terminal

```
>>> print('I am', 97, 'years old.')
I am 97 years old.
>>> print('a', 'b', 'c', sep='.', end='!')
a.b.c!>>>
```

Running Python
oooo

Processing
ooooooooooooooooooo

Input
ooo

Output
o●oooooo

THE format() FUNCTION

- accepts two arguments, a number, and a format specifier
- returns a formatted number as a string

Terminal

```
>>> format(12345.6789, '.2f')
'12345.68'
>>> format(12345.6789, '10,.2f')
' 12,345.68'
>>> format(12345, '8_d')
'  12_345'
```

FORMAT SPECIFIERS

basic form of a format specifier is:
'[width][grouping_option][.precision][type]'

- width: the minimum total length of string
- grouping_option: a character to use when grouping digits
- .precision: the number of digits after the decimal point
- type use d for integers, f for floats, % for percentages

Terminal

```
>>> format(12345.6789, '.5f')
'12,345.67890'
>>> format(12345, '12,d')
'      12,345'
>>> format(0.01, '0.0%')
'1%'
```

AN EXAMPLE PROGRAM

- statements execute in the order they appear

Editor - total_price.py

```python
1  # Ask User to input the item price
2  price = float(input('Enter the item\'s price: '))
3
4  # Ask User to input how many items
5  num_of_items = float(input('How many items: '))
6
7  # Calculate the total price
8  total_price = price * num_of_items
9
10 # Display the total price
11 print('The total price is $',
12        format(total_price, '.2f'), '.', sep='')
```

Running Python
○○○○

Processing
○○○○○○○○○○○○○○○○○○○○

Input
○○○

Output
○○○○●○○○

AN EXAMPLE PROGRAM CONTINUED.

Terminal

```
$ python total_price.py
Enter the item's price: .99
How many items: 8
The total price is $7.92.
```

Running Python
0000

Processing
00000000000000000

Input
000

Output
00000●00

F-STRINGS

- start with an f before the first quotation mark
- wrap variables in the string in braces { }
- add format specifiers inside the braces after a colon

Terminal

```
>>> cost = 12345.6789
>>> f'{cost}'
'12345.6789'
>>> f'The item costs ${cost}.'
'The item costs $12345.6789.'
>>> f'The item costs ${cost:.2f}.'
'The item costs $12345.68.'
>>>
```

Running Python
○○○○

Processing
○○○○○○○○○○○○○○○○○○○

Input
○○○

Output
○○○○○○●○

AN EXAMPLE PROGRAM - REVISITED

Editor - total_price.py

```python
1 price = float(input('Enter the item\'s price: '))
2 num_of_items = float(input('How many items: '))
3 total_price = price * num_of_items
4
5 # Display the total price
6 print(f'The total price is ${total_price:.2f}.')
```

Terminal

```
$ python total_price.py
Enter the item's price: .99
How many items: 8
The total price is $7.92.
```

Running Python
○○○○

Processing
○○○○○○○○○○○○○○○○○○○○

Input
○○○

Output
○○○○○○○●

Thanks for watching!