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

Lecture 01

Dr. John H. Cole
<jhcole@purdue.edu>

PURDUE
UNIVERSITY®

COLLEGE OF ENGINEERING

Fall 2022

Open a Terminal

- Linux & Mac

Terminal

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

- Windows (also called command prompt)

Terminal

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

- Generic Terminal

Terminal

```
$
```

Start Interactive Mode

Terminal

```
$ python
Python 3.10.5 (main, Jun 11 2022, 16:53:24)
[GCC 9.4.0] 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 typing `exit()` and pressing Enter
- statements are not saved

Interactive Mode - Example

Terminal

```
$ python
Python 3.10.5 (main, Jun 11 2022, 16:53:24)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license"
for more information.
>>> print('hello world')
hello world
>>> exit()
$
```

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
>>>
```

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

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

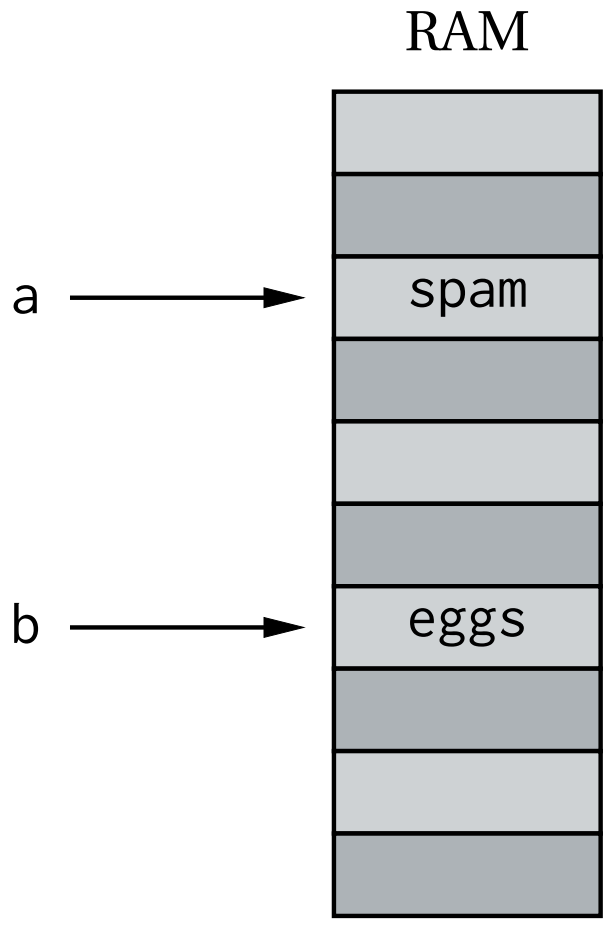
Assignment Statement sets the value that a variable refers to

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

Simple Assignment

Terminal

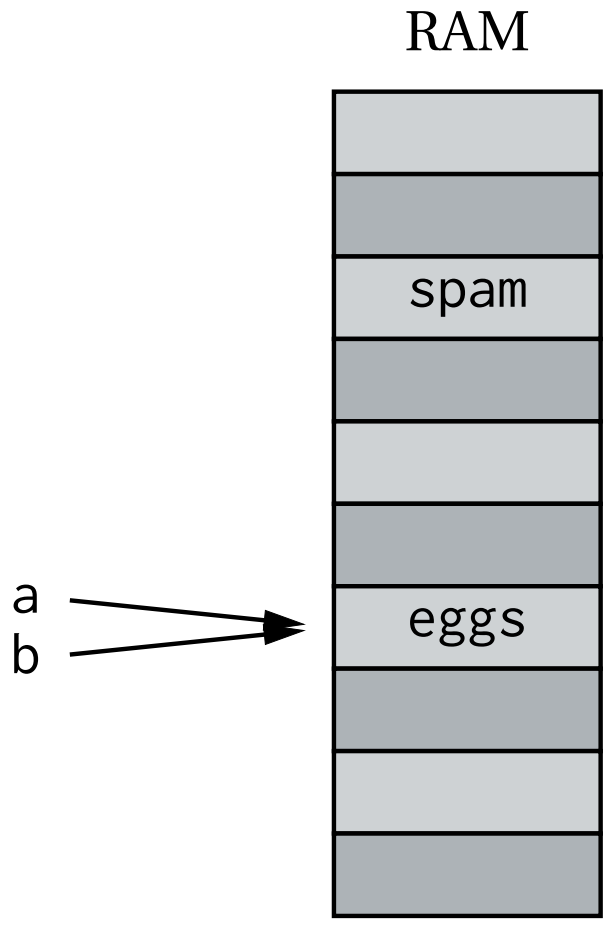
```
$ python
>>> a = 'spam'
>>> a
'spam'
>>> b = 'eggs'
>>> b
'eggs'
>>> a = b
>>> a
'eggs'
>>>
```



Simple Assignment

Terminal

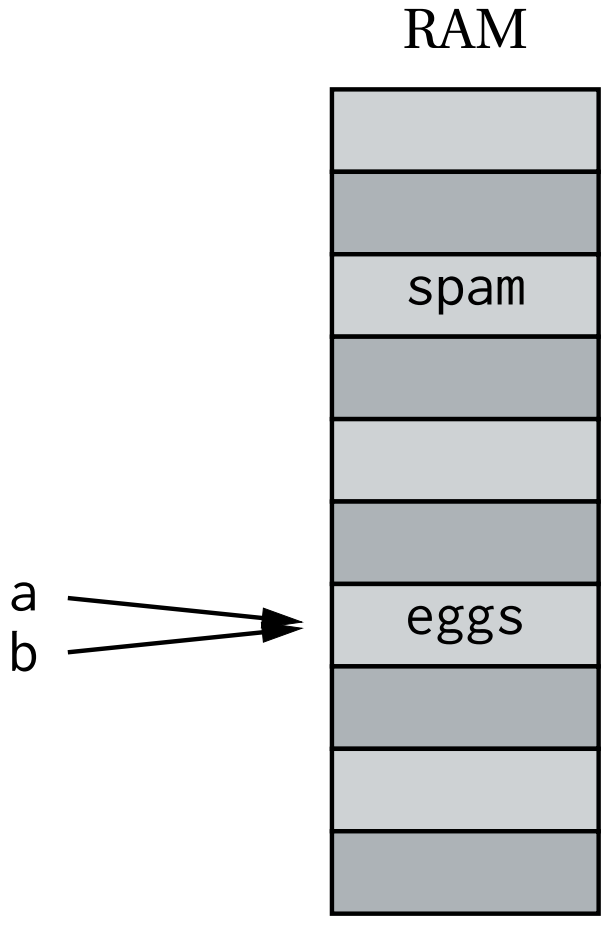
```
$ python
>>> a = 'spam'
>>> a
'spam'
>>> b = 'eggs'
>>> b
'eggs'
>>> a = b
>>> a
'eggs'
>>>
```



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



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

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'>
>>>
```

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

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

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'
```

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!>>>
```

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

```
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='')
```

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.
```

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.'
>>>
```

An Example Program - revisited

Editor - total_price.py

```
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.
```


Thanks for watching!