Data 301 Data Analytics
Python

Dr. Irene Vrbik

University of British Columbia Okanagan
irene.vrbik@ubc.ca

Why learn Python?

Python is increasingly the most popular choice of programming language for data analysts because it is designed to be simple, efficient, and easy to read and write.

There are many open source software and libraries that use Python and data analysis tools built on them.

We will use Python to learn programming and explore fundamental programming concepts of commands, variables, decisions, repetition, and events.

P.S. the name comes from Monty Python.

What is Python?

Python is a general, high-level programming language designed for code readability and simplicity.

Python is available for free as open source and has a large community supporting its development and associated tools.

Python was developed by Guido van Rossum and first released in 1991. Python 2.0 was released in 2000, and Python 3 was released in 2008.

Python 3 is backwards-incompatible meaning Python 3 code won't necessary run in Python 2.

Python Language Characteristics

Python supports:

- dynamic typing – types can change at run-time
- multi-paradigm – flexible in that is supports both procedural, object-oriented, and functional styles, for example.
- automatic memory management and garbage collection
- extensible – other languages such as C/C++ can be used to compile the code.
- and more ...

Python Language Characteristics

Python core philosophies (by Tim Peters)

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Some Quotes

If you can't write it down in English, you can't code it.

— Peter Halpern

I really hate this damn machine.
I wish that I could sell it.
I never does quite what I want.
Just only what I tell it.

— programmers lament

Introduction to Programming

Recall...

An *algorithm* is a precise sequence of steps to produce a result. A *program* is an encoding of an algorithm in a **language** to solve a particular problem.

There are numerous languages that programmers can use to specify instructions. Each language has its different features, benefits, and usefulness.

The goal is to understand fundamental programming concepts that apply to all languages.

Installing and Using Python

For this unit, we will be working with Python 3 (not Python 2).

Follow this Windows guide (or any of the other countless others you might find on the internet) on getting Python 3 on your computer.

If you are on a Mac, you should already have Python 2 installed; however, you'll need to update that to Python 3 with the Installer, or using Homebrew. Both techniques are summarized here.

Another way you can get python is through Anaconda (recommended). It includes some of most common data science packages and easy to work with.

- See this help video (Windows)
- See this help video (Mac)

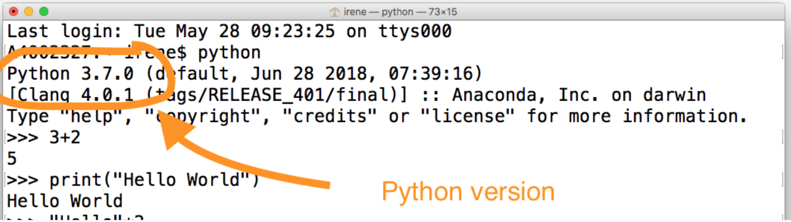
Installing and Using Python

Now that you've installed Python, you have the choice of entering the Python interpreter by either:

1. Command/Terminal Prompt
2. IDLE

For option 1. open the Command Prompt/Terminal, and type `python`[1] and press ENTER .

This will open up Python and print out your current version



```
Last login: Tue May 28 09:23:25 on ttys000
A4002327:~ irene$ python
Python 3.7.0 (default, Jun 28 2018, 07:39:16)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 3+2
5
>>> print("Hello World")
Hello World
```

[1] use `python -version` to check what version you're running

Installing and Using Python

IDLE is an integrated development environment (IDE) for Python. The Python installer for Windows contains the IDLE module by default and is an optional part of the Python packaging with many Linux distributions.

Assuming you have downloaded Python using Installer, in your applications folder you should see a Python3 folder and within it a program called IDLE.

In Windows, search for the IDLE icon in the start menu and double click on it.

This will open IDLE, where you can write Python code and execute it as just as you would in the console/terminal.

As a third option, you could one of the various online Python 3 compilers

Using Python

When you open Python you will see three Greater Than symbols
>>>

This is the so-called *prompt* indicating that you are now in an interactive Python interpreter session, also called the “Python shell" rather than normal terminal command prompt.

Let’s type some code for Python to run

Python example 1

```
>>> 3+2
5
>>> print("Hello, World!")
Hello, World!
>>>
```

Using Python

- The `help()` command is a useful function is used to get the documentation of specified module, class, function, variables etc. right from the interpreter. Press q to close the help window and return to the Python prompt. For example, to get help on the `print` function:

  Python help
  ```
  >>> help(print)
  ```

  Press q to close the help window and return to the Python prompt.

- To exit the Python shell in the console type Ctrl + Z the ENTER (on Windows) or Ctrl + D on a Mac. Alternatively, you could also run the python command `exit()`

Python: Basic Rules

- To program in Python you must follow a set of rules for specifying your commands. This set of rules is called a **syntax**.
- Just like any other language, there are rules that you must follow if you are to communicate correctly and precisely.
- For a more thorough overview of Python, you may have a look at Python Essential Reference by David Beazley as well as countless website online, eg w3schools, codeacademy, ...
- Other useful resources include: documentation for Python 3.7.3, Python.org (getting started), and the Pep 8 style guide.

Python: Basic Rules

Important general rules of Python syntax:

- Python *is* case-sensitive.
- Python *is* particular on whitespace and indentation.
- Use four spaces or tabs for indentation whenever in a block.
    - Spaces are the more “pythonic" indentation method.
    - Python 3 recognize either spaces or tabs but disallows mixing the two for indentation.

```
def spam():
    eggs = 12
    return eggs
print spam()
```

Comments

Comments are used by the programmer to document and explain the code. Comments are ignored by the computer.

Hence any characters appearing after the "#" are ignored by the computer.

Comment Example
```
# Single line comment
print(1)  # Comment at end of line
```

As suggested in the PEP8 Style Guide for Python Code inline comments (i.e. those appearing on the same line as a statement) should be used sparingly, should be separated by at least two spaces from the statement and should start with a # and a single space.

Python Programming

A Python program, like a book, is read left to right and top to bottom. Each command is on its own line.

The end of command is the end of line (i.e semi-colons are not a required terminator).

```
# Sample Python program
name = "Joe"
print("Hello")
print("Name: "+name)
```

The following would cause an error:

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

Python Programming

If you wanted to create a multi-line statement, we could do so using a backward slash

```
print("Hello\
World!")
```

Alternatively, we could wrap very long strings in triple quotes

```
print('''This is
a very long
string spanning
multiple lines ''')
```

Python Scripts

Rather than typing command directly into Python and running statements line by line, a user may type a Python program in a text document with the extension `.py`. These Python *scripts* and can be executed using the command[2]:

```
python PythonScriptName.py
```

[2]to be written in the Command prompt/terminal *not* in the Python shell

Python Scripts

For example if our file contained the following content:

```
print(3+2)
print("Hello World!")
4*8 # won't be printed without print statement
```

We would expect the following output:

```
ivrbik$ python PythonScriptName.py
5
Hello World!
```

Python Programming

You may decide to write your Python program in text editor specificly dedicated to Python (some examples) or an integrated development environment (IDE examples) instead of a basic editor like Notepad.

Some benefits of using an IDE/Python editor is that syntax is highlighting for ease of viewing, they may have code formatting capabilities (eg. shortcuts for commenting) and may have the ability to execute and debug code.

Python Editor - Jupyter

Another alternative is to create a *notebook* which is a single document that integrates both the code and its output.

Jupyter notebook is a graphical, browser-based application for editing and running Python. You can use it directly on your browser (i.e. without having to download anything) or you can install Notebook (recommended) by following the the instructions here.

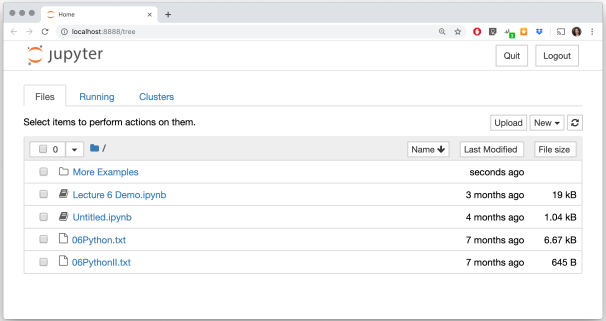
Once you have Python 3 and Jupyter notebook installed, you can run it by typing `jupyter notebook` in your Command Prompt.

This action will start the notebook server in your default browser and echo information about the notebook server in your terminal.

Read more about it here

Python Editor - Jupyter

The *Notebook Dashboard* will list all of the notebooks (.ipynb), files, and subdirectories stored in the *local* working directory (i.e. the directory from which you launched jupyter). You can navigate to a different directory by clicking though the filing system as you would in Terminal or a Windows machine.

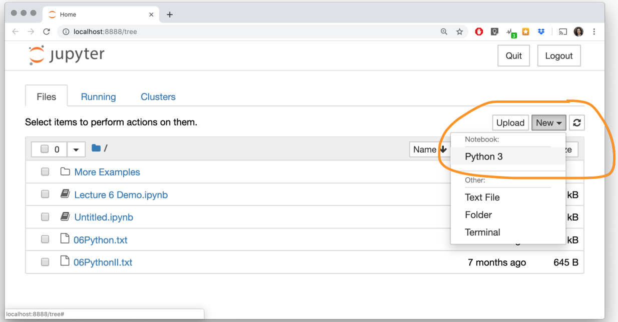


Select items to perform actions on them.

More Examples	seconds ago	
Lecture 6 Demo.ipynb	3 months ago	19 kB
Untitled.ipynb	4 months ago	1.04 kB
06Python.txt	7 months ago	6.67 kB
06PythonII.txt	7 months ago	645 B

Python Editor - Jupyter

To create a new notebook, select **New, Python3**.

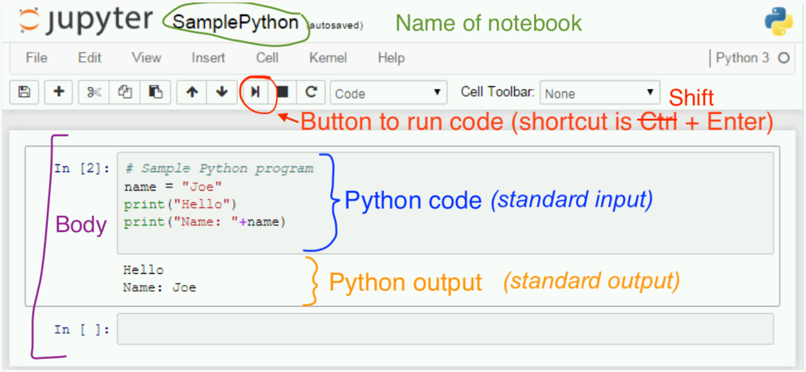


Select items to perform actions on them.

More Examples		
Lecture 6 Demo.ipynb		kB
Untitled.ipynb		kB
06Python.txt		kB
06PythonII.txt	7 months ago	645 B

Python Editor – jupyter notebook

You can change the default name from *Untitled* to something reasonable by double-clicking the title field.


jupyter
SamplePython (autosaved)
Name of notebook
File Edit View Insert Cell Kernel Help
Python 3
Code
Cell Toolbar: None
Shift
Button to run code (shortcut is ~~Ctrl~~ + Enter)
Body
Python code *(standard input)*
Python output *(standard output)*

```
In [2]: # Sample Python program
        name = "Joe"
        print("Hello")
        print("Name: "+name)

        Hello
        Name: Joe

In [ ]:
```

Python Editor – jupyter notebook

- The body of the notbook is comprised of *cells*:

  Markdown cells are used to build write regular (non-code) text. More markdown here and see helpful cheatsheet.

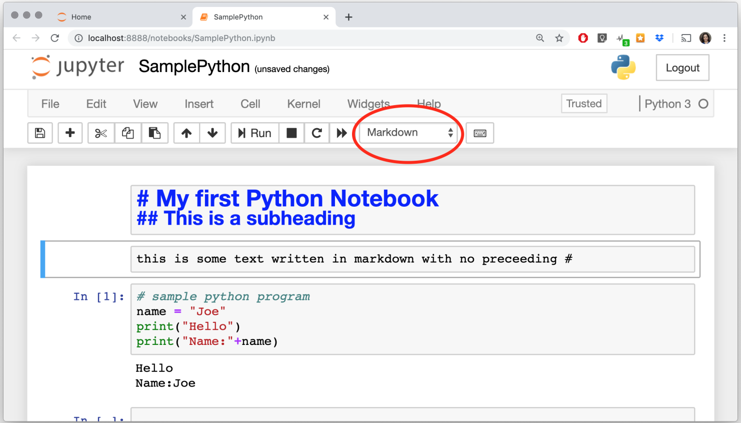
  Code cells (default) are used to define the code which will be compiled (after pressing ⏭ Run or pressing Shift + Enter) to produce output.

- You can inserts cells (either code/markdown) anywhere by clicking **Insert** > **Insert Cell Above** (or **Insert Cell Below**)

- Same thing goes for deleting/copying/pasting/undoing cells: see the **Edit** drop down menu bar.

Python Editor – jupyter notebook

You can create regular text and headings in “Markdown" mode. Headings are indicated using # (subheadings use ##)



```
# My first Python Notebook
## This is a subheading
```

```
this is some text written in markdown with no preceeding #
```

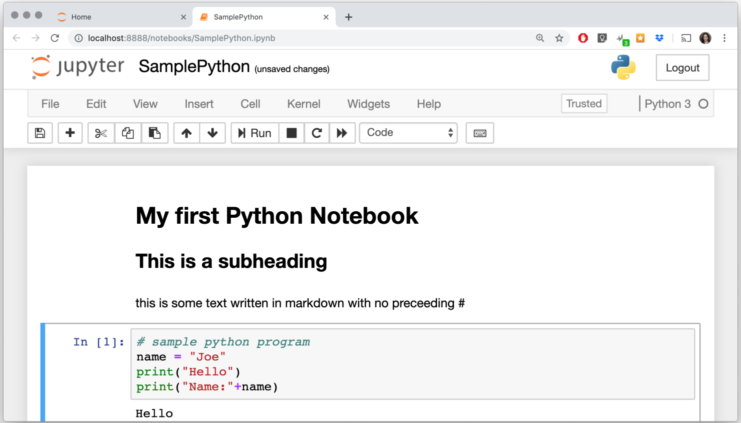
```
In [1]: # sample python program
        name = "Joe"
        print("Hello")
        print("Name:"+name)

        Hello
        Name:Joe

In [ ]:
```

Python Editor – jupyter notebook

- You can run a cell either by pressing Shift + ENTER or by pressing the little run button ▶| Run located at the top of the page (see *Help* $\rightarrow$ *Keyboard Shortcuts* for more)
- For markdown cells, Mardown code will be replaced by text will be formatted according to the Markdown language
- For code cells, Python will execute the code and place resulting output directly beneath the cell.
- By default, an empty code cell is automatically created at the bottom on the notebook.
- If you want to edit a cell, simply double click it.

Python Editor – jupyter notebook

This is what the markdown looks like after it is run:



My first Python Notebook

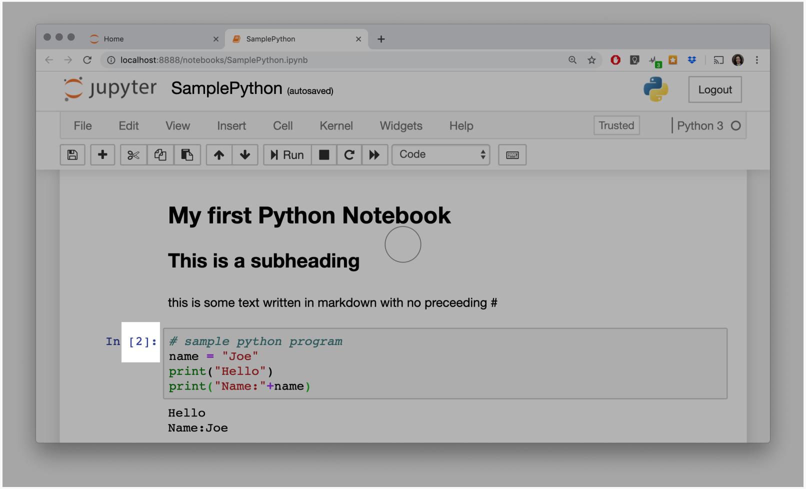
This is a subheading

this is some text written in markdown with no preceeding #

```
In [1]: # sample python program
        name = "Joe"
        print("Hello")
        print("Name:"+name)

        Hello
```

Python Editor – jupyter notebook

After a cell is run, a number will appear in the square parenthesis (an asterisk $*$ will appear for cells that are currently running). Every time we run a cell, this number (execution count) will increase.



My first Python Notebook

This is a subheading

this is some text written in markdown with no preceeding #

```
In [2]: # sample python program
        name = "Joe"
        print("Hello")
        print("Name:"+name)

        Hello
        Name:Joe
```

Python Editor – jupyter notebook

- While jupyter notebook does autosave periodically, it is a good to save your work upon exiting (go to **File** > **Save and Checkpoint** or click the save icon).
- This will automatically save a file with the name you provided at the top of the notebook with the extension .ipynb (eg SamplePython.ipynb)
- When we open the file again, it is not guaranteed that everything we need is in memory.
- It is good practice to “reset" the session by clicking **Kernel** > **Restart and Run all Cells** to ensure everything we need has been loaded into our working session (note that this will reset all of our numbers in the square brackets).

Python Editor - Jupyter

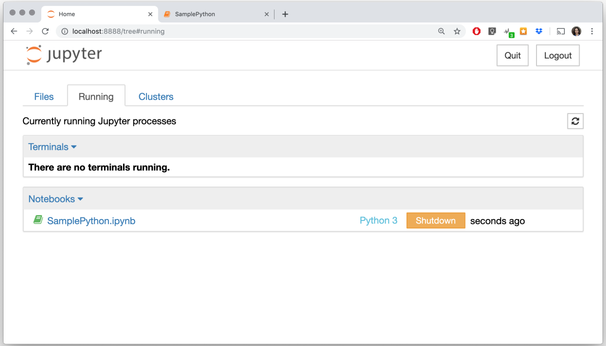
To close a notebook, navigate the the **Running** tab and click the orange Shutdown button or go to the associated notebook and click on menu *File* → *Close and Halt*

Closing the notebook's page is not sufficient to shutdown.

To close the program, you need to close the associated terminal using or press Ctrl + C in Terminal.

Closing the browser (or the tab) will not close the Jupyter Notebook App.

Python Editor - Jupyter



Currently running Jupyter processes

Terminals

There are no terminals running.

Notebooks

SamplePython.ipynb Python 3 Shutdown seconds ago

Python: Hello World!

A traditional introduction into programming involves outputting the message "Hello World!"

In Python3, this is a simple as typing:

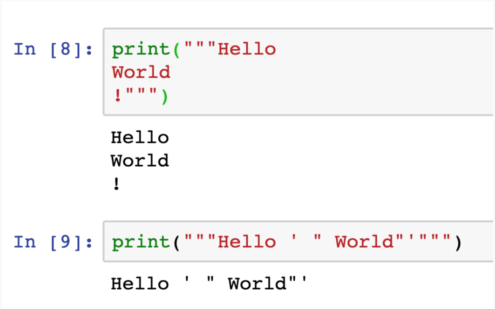
```
print("Hello World!")
```

The print function will print to the terminal (standard output) whatever data (number, string, variable) it is given.

You can use double quotes ("Hello world") or single quotes ('Hello world') just be consistent! E.g. 'Hello World!" will produce a error.

Tripple Quotes

- If your message runs across multiple lines, you can use 3 quotations to denote multi-line strings.
- The sting begins with a `'''` (or `"""`) and ends with a (or `"""`).
- Note that in this environment, we can use linebreaks (ie we can start a new line by pressing ENTER) and include the single and double quotes with our string.



```
In [8]: print("""Hello
        World
        !""")

        Hello
        World
        !

In [9]: print("""Hello ' " World"'""")

        Hello ' " World"'
```

Try It: Python Printing

Example 1

Write a Python program that prints "I can start coding!"

Example 2

Write a Python program that prints these three lines:

```
I know that I can program in Python.
I am programming right now.
My awesome program has three lines!
```

Example 3

How many of the following statements are TRUE?

1. Python is case-sensitive.
2. A command in Python must be terminated by a semi-colon.
3. Indentation does not matter in Python.
4. A single line comment starts with `"""`.
5. The print command prints to standard input.

A) 0 B) 1 C) 2 D) 3 E) 4

Answer:

How many of the following statements are TRUE?

1. Python is case-sensitive.
2. A command in Python must be terminated by a semi-colon.
3. Indentation does not matter in Python.
4. A single line comment starts with `"""`.
5. The print command prints to standard input.

A) 0 B) 1 C) 2 D) 3 E) 4

Answer:

How many of the following statements are TRUE?

1. Python is case-sensitive.
2. A command in Python must be terminated by a semi-colon.
3. Indentation does not matter in Python.
4. A single line comment starts with `""""`.
5. The print command prints to standard input.

A) 0 B) 1 C) 2 D) 3 E) 4

Answer:

How many of the following statements are TRUE?

1. Python is case-sensitive.
2. A command in Python must be terminated by a semi-colon.
3. Indentation does not matter in Python.
4. A single line comment starts with `"""`.
5. The print command prints to standard input.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Answer:

How many of the following statements are TRUE?

1. Python is case-sensitive.
2. A command in Python must be terminated by a semi-colon.
3. Indentation does not matter in Python.
4. A single line comment starts with `"""`.
5. The print command prints to standard input.

A) 0 B) 1 C) 2 D) 3 E) 4

Answer:

How many of the following statements are TRUE?

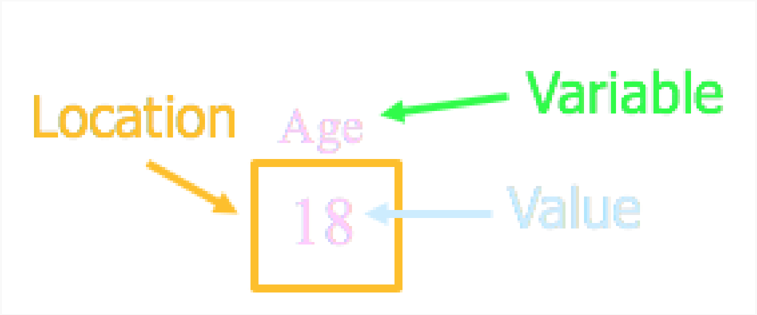
1. Python is case-sensitive.
2. A command in Python must be terminated by a semi-colon.
3. Indentation does not matter in Python.
4. A single line comment starts with `"""`.
5. The print command prints to standard input.

A) 0 B) *1* C) 2 D) 3 E) 4

Variables

Recall...

A *variable* is a name that refers to a location that stores a data value.



IMPORTANT: The *value* at a location can change using initialization or assignment.

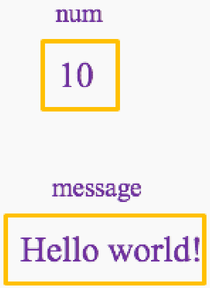
Variable Assignment

In Python the assignment operator is “=".

We will use it to (re)set value of a variable.

- Example:

```
num = 10
message = "Hello world!"
```



Python Variables

Variables are created when first assigned.

A variable type is dynamic and do not need to be declared.

It can store any particular type (e.g. int, float, strings, or Boolean) at any given time.

- Example:

```
val = 5
val = "Hello"
isAwesome = True
```

Recall: **Boolean** values can be either True or False (no quotes)
N.B. in Python case matters.

The type (string, int, float etc.) of the variable is determined by Python upon execution.

Variable Rules

Variables are a name that must begin with a letter or an underscore and cannot contain spaces. All subsequent characters must be letters, numbers or underscores.

Variables are created when they are first used. There is no special syntax to declare (create) a variable.

Variable names are case-sensitive.

A programmer picks the names for variables, but try to make the names meaningful and explain their purpose.

Variable Rules

- On top of the rules from the previous page, there are a number of reserved words you can't use in Python for variable names since they have special meaning in Python code.

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

- While these are the only rules, there are certain naming conventions outlined by PEP8 that you may choose to follow.

Example 4

How many of the following variable names are valid?

1. name
2. string2
3. 2cool
4. under_score
5. space name
6. else

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** ≥ 4

Answer:

How many of the following variable names are valid?

1. name
2. string2
3. 2cool
4. under_score
5. space name
6. else

A) 0 B) 1 C) 2 D) 3 E) ≥ 4

Answer:

How many of the following variable names are valid?

1. name
2. string2
3. 2cool
4. under_score
5. space name
6. else

A) 0   B) 1   C) 2   D) 3   E) ≥ 4

Answer:

How many of the following variable names are valid?

1. name
2. string2
3. 2cool
4. under_score
5. space name
6. else

A) 0 B) 1 C) 2 D) 3 E) ≥ 4

Answer:

How many of the following variable names are valid?

1. name
2. string2
3. 2cool
4. under_score
5. space name
6. else

A) 0 B) 1 C) 2 D) 3 E) ≥ 4

Answer:

How many of the following variable names are valid?

1. name
2. string2
3. 2cool
4. under_score
5. space name
6. else

A) 0 B) 1 C) 2 D) 3 E) ≥ 4

Answer:

How many of the following variable names are valid?

1. name
2. string2
3. 2cool
4. under_score
5. space name
6. else

A) 0 B) 1 C) 2 D) *3* E) ≥ 4

Python Math Expressions

Operation	Syntax	Example	Output
Add	+	5 + 3	8
Subtract	-	10 - 2	8
Multiply	*	5 * 3	15
Divide	/	8/4	2
Modulus	%	9 % 4	1
Exponent	**	5 ** 2	25

The modulo operation finds the remainder after division of one number by another (called the modulus of the operation). The modulus is 2 in the following examples (modulo is the operator %)

```
>>> 8%2         # even numbers should return 0
0
>>> 7%2         # odd numbers should return 1
1
```

Expressions - Operator Precedence

Each operator has its own priority similar to their priority in regular math expressions:

1. Any expression in parentheses is evaluated first starting with the inner most nesting of parentheses.
2. Exponents
3. Multiplication division and modulos (*, /, %)
4. Addition and subtraction (+,-)

Recall: **BEDMAS** *B*rackets *E*xponents *D*ivision, *M*ultiplication, (modulus), *A*ddition and *S*ubtraction
Example: `20 - ((4 + 5) - (3 * (6 - 2))) * 4` = 32

Python Expression Question

Example 5

What is the value of this expression

$$8 * *2 + 12/4 * (3 - 1)\%5$$

HINT: Divide, Multiply and Modulo are rank equally (and go left to right). See order of operations here.

A) 69 **B)** 65 **C)** 36 **D)** 16 **E)** 70

Python Expression Question

Answer:

What is the value of this expression

$$8 * *2 + 12/4 * (3 - 1)\%5$$

HINT: Modulo is executed after multiplication and division; more on this here.

A) 69 **B)** *65* **C)** 36 **D)** 16 **E)** 70

I think it's good practice to be explicit with brackets. I.e., I might have written the above as:

```
8**2 + ((12/4)*(3-1))%5
```

Try it: Python Variables and Expressions

Example 6

Write a program that prints the result of `35 + 5*10`

Example 7

Write a program that uses at least 3 operators to end up with the value 99.

Example 8

Write a program that has a variable called `name` with the value of *your* name and a variable called `height` storing your height in feet. Print out your name and height using these variables.

Rules for Stings in Python

Strings are sequences of characters that must be surrounded by single or double quotes.

The minimum number of characters is zero `""`, which is called the **empty string**.

Strings can contain most characters except enter, backspace, tab, and backslash. These special characters must be *escaped* by using the escape character: \

- Example:

  new line `\n`
  single quote `\'`
  backslash `\\`
  double quote `\''`

Strings

As mentioned previously, we can use triple quotes """ for a strings that contain single/double quotes and/or line breaks.

In addition, double quoted strings can contain single quoted strings and vice versa. Example:

```
name = 'Joseph "Joe" Jones'
storeName = 'Joe\'s Store'
storeName = "Joe's Store" # alternatively
height = '''5'9"'''
print("""String that is really long
with multiple lines
     and spaces is perfectly fine""")
```

Python String Indexing

Individual characters of a string can be accessed using square brackets ([]); the first character indexed at 0.

- Example:

```
str = "Hello"
print(str[1])  # e
print("ABCD"[0]) # A
print(str[-1]) # o
```

Negative values start at end and go backward, note that -1 yields the first number starting from the right. Read all more about strings here.

Example 9

How many of the following are valid Python strings?

1. `""`
2. `''`
3. `"a"`
4. `" "`
5. `"""`
6. `"Joe\' Smith\""`

A) 1 B) 2 C) 4 D) 5 E) 6

Answer:

How many of the following are valid Python strings?

1. `""`
2. `''`
3. `"a"`
4. `" "`
5. `"""`
6. `"Joe\' Smith\""`

A) 1 B) 2 C) 4 D) 5 E) 6

Answer:

How many of the following are valid Python strings?

1. `""`
2. `''`
3. `"a"`
4. `" "`
5. `"""`
6. `"Joe\' Smith\""`

A) 1 B) 2 C) 4 D) 5 E) 6

Answer:

How many of the following are valid Python strings?

1. `""`
2. `''`
3. `"a"`
4. `" "`
5. `"""`
6. `"Joe\' Smith\""`

A) 1 B) 2 C) 4 D) 5 E) 6

Answer:

How many of the following are valid Python strings?

1. `""`
2. `''`
3. `"a"`
4. `" "`
5. `"""`
6. `"Joe\' Smith\""`

A) 1 B) 2 C) 4 D) 5 E) 6

Answer:

How many of the following are valid Python strings?

1. `""`
2. `''`
3. `"a"`
4. `" "`
5. `"""`
6. `"Joe\' Smith\""`

A) 1 B) 2 C) 4 D) 5 E) 6

Answer:

How many of the following are valid Python strings?

1. `""`
2. `''`
3. `"a"`
4. `" "`
5. `"""`
6. `"Joe\' Smith\""`

A) 1 B) 2 C) 4 D) *5* E) 6

Python String Functions and Methods

Suppose:

```
st = "Hello"
st2 = "Goodbye"
```

Operation	**Command**	**Example**	**Output**
Length	`len()`	`len(st)`	`5`
Upper case	`upper()`	`st.upper()`	`HELLO`
Lower case	`lower()`	`st.lower()`	`hello`
Convert to a string	`str()`	`str(9)`	`"9"`
Concatenation	`+`	`st + st2`	`HelloGoodbye`

Substring	`[]`	`st[0:3]`
		`st[1:]`
			`Hel`
			`ello`

String to int	`int()`	`int("99")`	`99`

Dot Notation

- Like VBA, you will notice that Python uses the dot operator to perform *methods* on *objects* (read more here).
- Every constant, variable, or function in Python is actually a object with a type and associated attributes and methods.

```
>>> type("99")
<class 'str'>
>>> type(int("99"))
<class 'int'>
```

- A **method** is similar to a function however it is attached to an object (read more about this here)
- Note that we different syntax for functions vs. methods: For example `st.len` will produce an error as would `upper(st)`
- Here are some more examples of string methods.

String Operators: Concatenation

The *concatenation operator* is used to combine two strings into a single string. The notation is a plus sign "+".

- Example:

```
>>> st1 = "Hello"
>>> st2 = "World!"
>>> st3 = st1 + st2 # HelloWorld!
>>> print(st1+st1)
HelloHello
>>> print(st3)
HelloWorld!
```

String Operators: Concatenation

Note that we must hard code spaces if we want them:

```
>>> st4 = st1 +" "+ st2
>>> print(st4)
Hello World!
```

Concatenate with numbers and strings

We must convert numbers to strings using str before concatenation.

```
>>> num = 5
>>> print(st1+str(num))
Hello5
```

Python `print()`

Unlike with concatenation, we *can* mix types in the `print()` function.

In addition, notice how `print()` inserts spaces between inputs by default:

```
>>> print(st1,num, 100, "hi there", "byethere")
Hello 5 100 hi there byethere
```

We can change the default separator from `' '` to `', '` for example, as follows:

```
>>> print(st1,num, 100, "hi there", sep=", ")
Hello, 5, 100, hi there
```

String Operators: Deleting objects

If you have been following along with me, you will find that the code on the previous slide does not work.

This is because `str` is no longer treated as a function because I assigned `"Hello"` to this object on slide 52.

While `str` has a special meaning in Python, it is not a reserved word which will produce and error upon reassigning it a value in our program (i.e. it is not protected).

To make use of this function once more, we need delete the object we called `str` by typing:

```
del str
```

Example 10

What is the output of this code?

```
st1 = "Hello"
st2 = "World!"
num = 5
print(st1 + str(num) + "  " + st2)
```

A) Error

B) Hello5World!

C) Hello5 World!

D) Hello 5 World!

Answer:

What is the output of this code?

```
st1 = "Hello"
st2 = "World!"
num = 5
print(st1 + str(num) + "  " + st2)
```

A) Error

B) Hello5World!

C) *Hello5 World!*

D) Hello 5 World!

Substrings (slicing)

The substring function will return a range of characters from a string. The general syntax is `st[start:end]`

Substring indexing/slicing

- The `start` is inclusive the `end` is exclusive.
- If `start` is not provided, it defaults to 0.
- If `end` is not provided, it defaults to the end of the string.

- Example:

```
st = "Fantastic"
print(st[1])                    # a
print(st[0:6])         # Fantas
print(st[4:])         # astic
print(st[:5])          # Fanta
print(st[-6:-2])         # tast
```

Example 11

What is the output of this code:

```
st = "ABCDEFG"
print(st[1] + st[2:4] + st[3:] + st[:4])
```

A) ABCDCDEFGABCD

B) ABCDEFGABC

C) BCDDEFGABCDE

D) BCDDEFGABCD

E) BCDECDEFGABC

Answer:

What is the output of this code:

```
st = "ABCDEFG"
print(st[1] + st[2:4] + st[3:] + st[:4])
```

A) ABCDCDEFGABCD

B) ABCDEFGABC

C) BCDDEFGABCDE

D) *BCDDEFGABCD*

E) BCDECDEFGABC

Split

The *split* function will divide a string based on a separator. Without any arguments, it splits on whitespace,

```
>>> st = "Awesome coding! Very good!"
>>> print(st.split())
['Awesome', 'coding!', 'Very', 'good!']
```

otherwise is splits where ever it sees the inputted separator:

```
>>> print(st.split("!"))
['Awesome coding', ' Very good', '']
```

Split

This is very useful when we have, for example, comma separated values (csv):

```
>>> st = 'data,csv,100,50,,25,"use split",99'
>>> print(st.split(","))
['data', 'csv', '100', '50',
'', '25', '"use split"', '99']
```

Note that the returned object is a Python list.

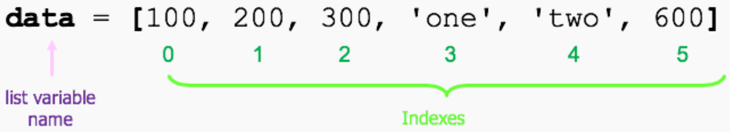
List Overview

A list is a collection of data items that are referenced by index.

- Lists in Python are similar to arrays in other programming languages

A list allows multiple data items to be referenced by one name and retrieved by index.

- Python list:



```
data = [100, 200, 300, 'one', 'two', 600]
```

Retrieving Items from a list

Items are retrieved by index (starting from 0) using square brackets:

```
data = [100, 200, 300, 'one', 'two', 600]
print(data[0])              # 100
print(data[4])              # 'two'
print(data[6])              # error ? out of range
print(data[len(data)-1]) # 600
print(data[-1])             # 600
print(data[2:4])            # [300, 'one']
```

You can create an empty list using:

```
emptyList = []
```

- To create a list with a range of numbers we can use range:

```
l = list(range(start, stop, step))
```

having parameter values:

  - start: Optional. An integer number specifying at which position to start. Default is 0
  - stop: Required. An integer number specifying at which position to end.
  - step: Optional. An integer number specifying the incrementation. Default is 1

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print(list(range(10)))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print(list(range(1,10)))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print(list(range(1,10,2)))
[1, 3, 5, 7, 9]
```

List manipulation

We can modify a single value by use of indices and the assignment operator =

```
>>> data = [1,2,3,5]
>>> data[2] = 7
>>> print(data)
[1, 2, 7, 5]
```

We can also modify multiple values at a time in the following way:

```
>>> data[0:1] = ["one","two"]
>>> print(data)
['one', 'two', 2, 7, 5]
```

Appending to a list

- Notice that when we try to add a value to the end of the list, we get an error:

```
>>> data[5] = 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

- To add an item to the end of the list, use `append()`.

Appending to a list

- Notice how `append()` doesn't return a new list; rather it *modifies* the original list.

```
>>> data = [1,2,3,5]
>>> data.append(7)
>>> print(data)
[1, 2, 3, 5, 7]
```

- Alternatively we could have used

```
>>> data = [1,2,3,5]
>>> data = data + [7]
>>> data
[1, 2, 3, 5, 7]
```

List in loops

- We can iterate over a list in a `for` loop.

```
colours = ['red', 'yellow', 'green', 'blue']
for colour in colours:
    print(colour)
```

- The following code appends the values in `i` one-by-one to the empty list `j`.

```
i = [1, 2, 3, 5, 8, 13]
j = []
for l in i:
    j.append(l)
```

List append vs extend

- append() adds its argument as a *single* element (eg. a number, a string or a another list) to the end of an existing list.

```
>>> x = [1, 2, 3] # len(x) = 3
>>> x.append([4,5])
>>> x
[1, 2, 3, [4, 5]]
>>> len(x)
4
```

- Notice how the length of the list increases by one.
- If we want to add the elements one-by-one to the end of x we could either use a loop as we did in the previous slide of use the extend() function.

List extend

- extend() iterates over its argument and adds each element to the list thereby *extending* the list.
- The length of the list increases by number of elements in it's argument.

```
>>> x = [1, 2, 3] # len(x) = 3
>>> x.extend([4,5])
>>> x # len(x) = 5
[1, 2, 3, 4, 5]
```

- Alternatively, we could have used

```
>>> x = [1, 2, 3] # len(x) = 3
>>> x = x + [4,5]
>>> print(x)
[1, 2, 3, 4, 5]
```

List extend vs append example

Read more about the difference between the two here

```
>>> x = [1, 2, 3]
>>> x.append([4, 5])
>>> print (x)
[1, 2, 3, [4, 5]]
>>> x[3]
[4, 5]
```

where `x[3]` returns `[4, 5]` and `x[4]` returns an error.

```
>>> x = [1, 2, 3]
>>> x.extend([4, 5])
>>> print (x)
[1, 2, 3, 4, 5]
>>> x[3]
4
```

List Operations

If `data = [1, 2, 3, 5]` and `lst = []`

Operation	Syntax	Examples	Output
Add item	<list>.append(val)	data.append(1)	[1, 2, 3, 5, 1]
Insert item	<list>.insert(idx,val)	data.insert(3,4)	[1, 2, 3, 4, 5]
Remove item	<list>.remove(val)	data.remove(5)	[1, 2, 3]
<list>.pop(ind)	data.remove(0)	[2, 3,5]	
Update item	list[idx]=val	data[0]=10	[10, 2, 3, 5]
Length of list	len(<list>)	len(data)	4
Slice of list	list[x:y]	data[0:3]	[1, 2, 3]
Find index	<list>.index(val)	data.index(5)	3
Sort list	<list>.sort()	data.sort()[3]	[1, 2, 3, 5]
Add	lst = []	lst.append(1)	[1]

See more here

[3] To sort in reverse order, use `data.sort(reverse=True)`

List details

It was mentioned already but its worth repeating...

For loops that are used to iterate though items in a list:

```
data = [5,9,-2,9]
for v in data:
   print(v)

 #### output:
# 5
# 9
# -2
# 9
```

List details

Note that this is not restricted to numbers:

```
data = ["apples", "bananas","oranges"]
for v in data:
    print(v)
```

prints `apples`, `bananas`, `oranges` (each on a separate line).

We could even iterate through characters in a string:

```
for v in "bananas":
    print(v)
```

prints `b`, `a`, `n`, `a`, `n`, `a`, `s` (each letter on a separate line)

List details

If we want to iterate through both index and value, we could use the `enumerate()` function.

```
data = ['apple', 'banana', 'grapes', 'pear']
for c, value in enumerate(data):
    print(c, value)

# Output:
# 0 apple
# 1 banana
# 2 grapes
# 3 pear
```

List details

If we want our index to start at 1 rather than 0, we could specify that as the second argument: `enumerate()` function.

```
data = ['apple', 'banana', 'grapes', 'pear']
for c, value in enumerate(data, 1):
    print(c, value)

# Output:
# 1 apple
# 2 banana
# 3 grapes
# 4 pear
```

Advanced: Python List Comprehensions

List comprehensions build a list using values that satisfy a criteria.

- Example:
  `evenNums100 = [n for n in range(101) if n%2==0]`
- Equivalent to:

```
    evenNums100 = []
    for n in range(101):
        if n%2==0:
            evenNums100.append(n)
```

```
# another example:
>>> squares = [x**2 for x in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Advanced: Python List Slicing

List slicing allows for using range notation to retrieve only certain elements in the list by index. Syntax:

```
list[start:end:step]
```

- Example:

```
data = list(range(1,11))
print(data) # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(data[1:8:2]) # [2, 4, 6, 8]
print(data[1::3]) # [2, 5, 8]
```

Example 12

At what index is item with value 3?

```
data = [1, 2, 3, 4, 5]
data.remove(3)
data.insert(1, 3)
data.append(2)
data.sort()
data = data[1:4]
```

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** not there

Answer:

At what index is item with value 3?

```
data = [1, 2, 3, 4, 5]
data.remove(3)
data.insert(1, 3)
data.append(2)
data.sort()
data = data[1:4]
```

A) 0 B) 1 C) *2* D) 3 E) not there

```
[2,2,3]
```

Try it: Lists

Example 13 (Question 1)

Write a program that puts the numbers from 1 to 10 in a list then prints them by traversing the list.

Example 14 (Question 2)

Write a program that will multiply all elements in a list by 2. Bonus: try doing this using the `enumerate()` function.

Example 15 (Question 3)

Write a program that reads in a sentence from the user and splits the sentence into words using split(). Print only the words that are more than 3 characters long. At the end print the total number of words.

Lists

In the previous example we use the `list()` function to create our list instead of the square brackets.

This constructs a list using the single input. This could be

- a sequence (eg. string, tuples) or
- a collection (set, dictionary) or
- a iterator object (like the objects iterated over in our `for` loops)

If no parameters are passed, it creates an empty list.

Tuples

- A tuple is a collection which is ordered and unchangeable. To create tuples we use round brackets `()`.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

- Elements in a tuple are referenced the same way as lists:

```
>>> print(thistuple)
('apple', 'banana', 'cherry')
>>> thistuple[1]
'banana'
```

- Tuples are useful for storing some related information that belong together.

Tuples

Unlike list, once a tuple is created, values can not be changed.

```
>>> thistuple[1] = "pineapple"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Notice that tuples are also iterable, meaning we can traverse through all the values. eg,

```
for i in thistuple:
   print(i)
```

The above prints:

```
apple
banana
cherry
```

Python Sets

- A set (like a mathematical set) is a collection which is unordered and unindexed.
- Sets are written with curly brackets {}.

```
>>> # N.B they do not carry duplicates
>>> thisset = {"apple", "banana", "cherry", "apple"}
>>> print(thisset)
{'apple', 'banana', 'cherry'}
```

- Since sets are unordered, the items will appear in a random order and elements cannot be reference by index.
- Again we can iterate though each item using a `for` loop.
- Read more about these here.

Python Dictionary

A dictionary is a collection which is unordered, changeable and indexed. We create them with curly brackets and specify their keys and values.

```
>>> thisdict = {
...   "key1": "value1",
...   "key2": "value2",
...   "key3": "value3"}
>>> print(thisdict)
{'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
```

N.B Keys should be unique. If you create different values with the same key, Python will just overwrite the value of the duplicate keys.

```
>>> nono = {'key1':'this', 'key1':'that'}
>>> nono
{'key1': 'that'}
```

Python Dictionary

We can now reference elements by a given *name* (i.e key) rather than the standard integers index.

```
>>> thisdict['key1']
'value1'
```

Referencing by index won't work (remember these are unordered)

```
>>> thisdict[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
```

Don't get confused at try to do indexing using the values rather than the keys! For instance the following would produce an error:

```
>>> thisdict['value1']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'value1'
```

Python Dictionary

If we wanted to, we could always use integers for the keys (in that case we don't need quotes around them when indexing using square brackets).

```
dict = {1:'one', 2:'two', 3:'three'}
print(dict[1])          # one
print(dict['one'])      # error - key not found
if 2 in dict:           # check if key exists
    print(dict[2])      # 'two'
```

We can add/delete and view keys/values using the following:

```
dict[4] = 'four'  # Add 4:'four'
del dict[1]       # Remove key 1
dict.keys()       # Returns keys
dict.values()     # Returns values
```

Python Dictionary

Iterating over a dictionary will return the *keys*

```
>>> dict = {'Martha':4, 'Janet':17, 'Peter':10}
>>> dict['Anna'] = 20
>>> for i in dict:
...     print(i)
...
Martha
Janet
Peter
Anna
```

To get the values use:

```
>>> for i in dict: print(dict[i]) # or
>>> for i in dict.values(): print(i)
```

Python Dictionary

To access both keys and values use:

```
# option 1:
# i = key, dict[i] = value
for i in dict:
    print("key = ", i, "value = ", dict[i])
```

```
# option 2:
for key, val in dict.items():
    print("key = ", key, "values =", val)
```

Output:

```
key =  Martha value =  4
key =  Janet value =  17
key =  Peter value =  10
key =  Anna value =  20
```

Summary of Python Structures

Lists can be altered and hold different data types:

```
lectures = [1,2,['excelI',excelII'],'CommandLine']
 # can delete individual items
del lectures[3]
 # reassign values
lectures[1] = 'Introduction'
print(type(lectures[0]))  # <class 'int'>
print(type(lectures[1]))  # <class 'str'>
print(type(lectures[2]))  # <class 'list'>
```

Tuples are *immutable* (we can't change the values):

```
topics= (1,2,['excel1','excel2'],'CommandLine')
del topics[3]
# TypeError: 'tuple' object doesn't support item deletion
type(topics[2])
# <class 'list'>
```

Summary of Python Structures

Sets do not hold duplicate values and are unordered.

```
>>> myset={3,1,2,3,2}
>>> myset
{1, 2, 3}
>>> myset[1]
TypeError: 'set' object does not support indexing
```

Dictionaries hold key-value pairs (just like real life dictionaries hold word-meaning pairs).

```
>>> wordoftheday = {
 'persiflage':'light, bantering talk or writing' ,
 'foment':'to instigate or foster'}
>>> wordoftheday['foment']
'to instigate or foster'
>>> wordoftheday[1] # produces an error
```

Example 16

What value is printed?

```
data = {'one':1, 'two':2, 'three':3}
data['four'] = 4
sum = 0
for k in data.keys():
    if len(k) > 3:
        sum = sum + data[k]
print(sum)
```

A) 7 **B)** 0 **C)** 10 **D)** 6 **E)** error

Answer:

At what index is item with value 3?

```
data['four'] = 4
sum = 0
for k in data.keys():
    if len(k) > 3:
        sum = sum + data[k]
print(sum)
```

A) *7*   B) 0   C) 10   D) 6   E) error

Try it: Dictionary

Example 17

Write a program that will use a dictionary to record the frequency of each letter in a sentence. Read a sentence from the user then print out the number of each letter.

- Code to create the dictionary of letters:

```
import string
counts = {}
for letter in string.ascii_uppercase:
    counts[letter] = 0
print(counts)
```

Example 18

Create the following two variables and write a Python program that prints the following output:

```
name = "Joe"
age = 25
```

output

```
Name: Joe
Age: 25
```

Example 19

Define the variable:

```
name = "Steve Smith"
```

Use substring to write a Python program that prints out the first name and last name of `Steve Smith` like below.

output
```
First Name: Steve
Last Name: Smith
```

Bonus challenge: Recode so that it would work with any name.

Print Formatting

One of the most obvious changes between Python 2 and Python 3 is how they use `print`:

```
print "Hello"
```

and Python 3:

```
print("Hello")
```

In Python3, the `print` method can accept parameters for formatting. See some examples on the next slide . . .

Print Formatting

```
print("Hi", "Amy", ", your age is", 21)
print("Hi {}, your age is {}".format("Amy",21))
print("Hi {1}, your age is {0}".format(21,"Amy"))
print("Hi {name}, your age is {age}".format(age=21,
name="Amy"))
```

- We can think of `{}` as placeholder arguments for the inputs given in `format(<input0>, <input1>)`.
- By default, these inputs will be added in the string in order (the `input0` will appear in the first place holder, `input1` in the second place holder, etc.
- If we want to read `input1` before `input0`, we need to refer to it by its integer index via `{1}` or by its name if we have provided one `{age}`

Python Modules

A Python *module* or library is code written by others for a specific purpose. Whenever coding, make sure to look for modules that are already written for you to make your development faster! Modules are imported using the import command:

```
import <modulename>
```

Useful modules for data analytics:

- `Biopython` (bioinformatics),
- `NumPy` (scientific computing/linear algebra),
- `scikit-learn` (machine learning), `pandas` (data structures),
- `BeautifulSoup` (HTML/Web)

Python Modules

- The general syntax to import the module named `mymodule` is:

```
import mymodule
```

- You can choose to import only parts from a module, by using the `from` keyword. For example, if we just want the object `person1` from `mymodule` type:

```
from mymodule import person1
```

- To import all functions/constants/object from `mymodule` use:

```
from mymodule import *
```

- Read more about modules here and `import` here.

Python: more on `import`

- When you use `import mymodule`, you import the module by the name of `mymodule`.
- To access any a particular attribute (eg. object or function) of that module, you need to define a complete model path using the dot notation.
  - For instance, if we wanted to access `person1` from `mymodule` you would need to use `mymodule.person1`.
- When you use "`from mymodule import person1`", the module is not imported, rather just `person1` has been imported as a variable.
  - We can now access it simply using `person1`.

Python Module

importing modules

```
import math # imports 'math' (of class 'module')
# access pi using dot notation
print(math.pi)
print(math.factorial(6))
print(pi) # NameError: name 'pi' is not defined

from math import pi
# access pi directly
print(pi)
factorial(6) # NameError: name 'factorial' is not defined

from math import *
# access pi (and anything else in the module) directly
print(pi)
print(factorial(6))
```

Python Modules

- There are a number of packages are automatically installed into the Python Standard Library.
- To install a module not in this standard library, use `pip`.
- The general syntax for install a module in the Command Prompt/Terminal (*not* the Python interpreter) is:

```
pip install <module_name>
```

- Alternatively, if you installed Anaconda you can use:

```
conda install <module_name>
```

- Click here to read about the differences between `pip` & `conda`.

Importing Objects from a Module

- For example, Python supports date and time data types and functions. To use, import the `datetime` *module* using the following:

```
import datetime
```

- We may choose to only import the `datetime` object from the `datetime` module (which happens to be the same name) using the following:

```
from datetime import datetime
```

Python Date and Time

The `datetime` object has a method for formatting date objects into readable strings. Read more here.

```
now = datetime.now()
>>> now = datetime.now()
>>> print(now)
2018-10-07 12:31:43.830464
>>> current_year = now.year
>>> current_month = now.month
>>> current_day = now.day
>>> print("{}-{}-{} {}:{}:{}".format(now.year, now.month,
now.day, now.hour, now.minute, now.second))
2018-10-7 12:31:43
>>> print("{}-{}-{} {}:{}:{}".format(now.year, now.month,
now.day, now.hour, now.minute, now.second))
2018-10-7 12:31:43
```

Python Clock

- The `time` module, is another useful package for handling time-related tasks.
- The `time()` function for example, returns the current time in seconds.
- On Linux machines, this is an integer counting the number of seconds passed since January 1, 1970, 00:00:00 (recall from Lecture 2).
- This function can be useful when we want to time how long a process takes within our program.

Python Clock

- Example:

```
>>> import time
>>> startTime = time.time()
>>> print("Start time:", startTime)
Start time: 1538941011.206657
>>> print("How long will this take?")
How long will this take?
>>> endTime = time.time()
>>> print("End time:", endTime)
End time: 1538941011.2094998
>>> print("Time elapsed:", endTime-startTime)
Time elapsed: 0.0028429031372070312
```

Python Input

To read from the keyboard (standard input), use `input`:

the input() function

```
 # input some text
name = input("What's your name?")

 # input an integer
age = input("What's your age?")
age + 10
TypeError: can only concatenate str (not "int") to str
int(age) + 10

 # input a float
num = input("What is 1 divided by 4?") # string
num = float(num) # reassign the num variable
 # all in one line
num = float(input("What is 1 divided by 4?"))
```

Python Input

- When your program executes an `input()` command the program will be stopped until the user has provided input.
- The `input()` function has an optional `prompt` argument. Usage: `input("optional prompt message")`
- This text will be displayed on the output screen upon execution and can provide the user some instruction on what to type.
- Whatever the user enters as input, the `input` function will convert into a string. You can convert this using typecasting
  - If you enter an integer value you need to explicitly convert it into an integer in your code using `int()`
  - If you enter an float value you need to explicitly convert it into an integer in your code using `float()`

Try it: Python Input, Output, and Dates

Example 20

Write a program that reads a name and prints out the name, the length of the name, the first five characters of the name.

Example 21

Print out the current date in `YYYY/MM/DD` format.