YOUR GUIDE TO THE PYTHON print() FUNCTION

What you will learn:

1. Basic usage and string literals
2. String Formatting
3. “sep”, “end” and “flush”
4. Printing to files/streams
5. Custom data types
6. Python 2 “print” vs Python 3 “print()”
7. Pretty print
8. ANSI Escape Sequences
9. Simple Animation
10. Mocking print() in unit tests
11. Debugging
12. Related information

PYTHON 2 vs 3

Warning: print() has changed between Python 2 and 3. This course covers the differences but otherwise the examples use Python 3 syntax

BASIC USAGE

```
>>> print()

>>> print("Hello world!")
Hello world!
>>> print("What is the answer?", 42)
What is the answer? 42
```

STRINGS

- Strings store text in your code
- There are three ways of declaring a string:

```
"Between double quotes"

'Between single quotes'

"""Using triple-quotes for
strings with multiple
lines"""
```

- Python style guide says to be consistent
- Triple-quotes are often used for documentation strings

STRINGS

- Supporting two kinds of quotes makes putting a quote in a string easier

```
"This sentence doesn't have a contraction"

'My favourite quote is: "Tis but a scratch" - Black Knight'

'<div class="md-col">'
```

QUOTES

```
>>> "Using double quotes"
'Using double quotes'
>>> 'Using single quotes'
'Using single quotes'
>>> """Using triple
... quotes for multiple
... lines"""
'Using triple\nquotes for multiple\nlines'
```

NEW LINE CHARACTER

```
>>> """Using triple
... quotes for multiple
... lines"""
'Using triple\nquotes for multiple\nlines'
```

- “\n” denotes moving to the next line
- Different operating systems deal with this differently, but Python takes care of that for you
- By default, print() automatically adds a “\n” to the end of each string before printing it

OTHER "\" CHARACTERS

`\n`	Newline
`\t`	Tab
`\a`	ASCII Bell
`\ooo`	Character with octal value "ooo"
`\xhh`	Character with hex value "hh"

`\\`	Backslash
`\"`	Double quote
`\'`	Single quote
`\b`	Backspace
`\r`	Carriage return

ESCAPE SEQUENCES

```
>>> print("First line \nSecond line \n\tThird line with tab")
First line
Second line
        Third line with tab
>>> print("Bong \a")
Bong
>>> print("Octal 275 is a half: \275")
Octal 275 is a half: ½
>>> print("Hex 0xBD ix Octal 275: \xBD")
Hex 0xBD ix Octal 275: ½
>>> print("Embedding quotes in quotes \" or \' ")
Embedding quotes in quotes " or '
>>> print("What if we want a backslash? \\ ")
What if we want a backslash? \
>>> print("Carriage return means go back \r to the start of the line")
 to the start of the lineback
```

PATHS AND REGEXES

- Backslash means escape, so text with backslashes in it can be painful
  - Windows paths: `C:\Windows\System32`
  - Regular Expressions: `^\d+(\.\d+)?`

```
“C:\\Windows\\System32”
“^\\d+(\\.\\d+)?”
```

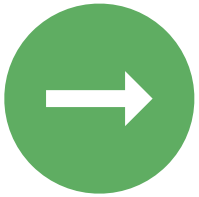
- Pre-fix your string with “r” to make it “raw” -- escape free

```
r“C:\Windows\System32”
r“^\d+(\.\d+)?”
```

MULTIPLICATION

```
>>> print(70*'=')
======================================================================
```

NEXT UP...



Different ways to format strings

TABLE OF CONTENTS

FORMATTING STRINGS

- Two strings can be concatenated together using “+”

```
“Hello “ + “World!”
```

- There are three ways of formatting strings in Python

```
'Hello %s' % name    # C-style

'My name is {1}, {0} {1}'.format('James', 'Bond')  # added in Python 3.0

f'I am {age} years old'  # f-string added in Python 3.6
```

STRING FORMATTING

```
>>> first = 'James'
>>> last = 'Bond'
>>> age = 90
>>> message = 'No, Mr. %s, I expect you to die' % last
>>> print(message)
No, Mr. Bond, I expect you to die
>>> print('The names is %s, %s %s' % (last, first, last))
The names is Bond, James Bond
>>> print('Sean Connery is now %d years old' % age)
Sean Connery is now 90 years old
>>> print('pi: %f \nshort pi %0.2f' % (math.pi, math.pi))
pi: 3.141593
short pi 3.14
>>> message = f'No, Mr. {last}, I expect you to die'
>>> print(message)
No, Mr. Bond, I expect you to die
>>> print(f'The name is {last}, {first} {last}')
The name is Bond, James Bond
>>> print(f"Sean's age times pi is {age*math.pi}")
Sean's age times pi is 282.7433388230814
>>> print(f"Sean's age times pi is {age*math.pi:.2f}")
Sean's age times pi is 282.74
```

MORE INFO

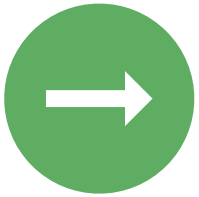
- Full lessons are available on string formatting on RealPython:
- Text article:

https://realpython.com/python-f-strings/

- Video tutorial:

https://realpython.com/courses/python-3-f-strings-improved-string-formatting-syntax/

NEXT UP...



The “sep”, “end” and “flush” arguments to print() and how they control the appearance of your output

TABLE OF CONTENTS

sep

```
>>> print('There are', 6, 'members of Monty Python')
There are 6 members of Monty Python
>>> message = 'There are' + 6 + 'members of Monty Python'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>> message = 'There are' + str(6) + 'members of Monty Python'
>>> print(message)
There are6members of Monty Python
>>> print('There are', 6, 'members of Monty Python', sep='😀')
There are😀6😀members of Monty Python
>>> print('There are', 6, 'members of Monty Python', sep=' ')
There are 6 members of Monty Python
>>> print('There are', 6, 'members of Monty Python', sep=None)
There are 6 members of Monty Python
>>> print('There are', 6, 'members of Monty Python', sep='')
There are6members of Monty Python
>>> print('There are', 6, 'members of Monty Python', sep='\n')
There are
6
members of Monty Python
>>> data = [
...     ['year', 'last', 'first'],
...     [1943, 'Idle', 'Eric'],
...     [1939, 'Cleese', 'John']
... ]
>>> for row in data:
...     print(*row, sep=',')
...
year,last,first
1943,Idle,Eric
1939,Cleese,John
```

NAMED ARGUMENTS

- Python allows multiple unnamed arguments

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- The "***objects**" argument is accessible as a list inside of the function
- As there are an unknown number of items passed in, the remaining parameters must be named explicitly when calling the function

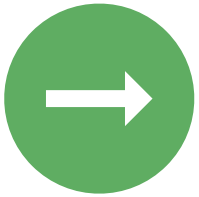
end AND flush

```
import time

def count_items(items):
    print('Counting ', end='', flush=True)
    num = 0
    for item in items:
        num += 1
        time.sleep(1)
        print('.', end='', flush=True)

    print(f'\nThere were {num} items')
```

OTHER USES

```
def sentences():
    print('The first sentence', end='. ')
    print('The second sentence', end='. ')
    print('The last sentence.')

def planets():
    print('Mercury', 'Venus', 'Earth', sep=',', end=', ')
    print('Mars', 'Jupiter', 'Saturn', sep=',', end=', ')
    print('Uranus', 'Neptune', sep=',', end=', ')

def bullets():
    print('My favourite Pythons:', end='\\n *')
    print('Idle', end='\\n *')
    print('Cleese ', end='\\n *')
    print('Sumatran short-tailed')
```

NEXT UP...



Printing to more than just the screen

TABLE OF CONTENTS

FILE STREAMS

- print() wraps the system calls that do the actual printing to the screen
- Operating Systems have three standard streams:
  - <stdin>, <stdout>, <stderr>

```
>>> import sys
>>> sys.stdin
<_io.TextIOWrapper name='<stdin>' mode='r' encoding='UTF-8'>

>>> sys.stdout
<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>

>>> sys.stderr
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>
```

WRITING WITH SYS

```
>>> import sys
>>> result = sys.stdout.write('hello\n')
hello
>>> result
6
```

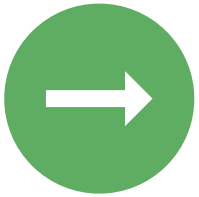
FILE OBJECTS

- print()'s "file" parameter points to the file-object it will print to, it defaults to <stdout>

```
with open('file.txt', mode='w') as file_object:
    print('hello world', file=file_object)
```


Warning: <stdout> and print() only handle string data, writing binary will not work

NEXT UP...



Customizing your classes so they better interact with print()

TABLE OF CONTENTS

DEFINING str AND repr

- print() converts its arguments to strings using str()
- str() looks for one of two methods on a class for converting to a string:
  - __str()__
  - __repr()__
- If neither is defined, str() uses the built-in default
- “__str()__” should be used for human readable content

USING repr

- Like “str()” there is also a “repr()” method, it only looks for “__repr()__”
- Python documentation says:

> For many types, this function makes an attempt to return a string that would yield an object with the same value when passed to `eval()`, otherwise the representation is a string enclosed in angle brackets that contains the name of the type of the object together with additional information often including the name and address of the object..

CUSTOM OBJECTS

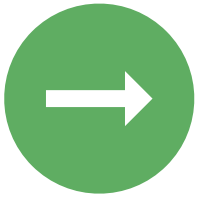
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f'Person({self.name})'

    def __repr__(self):
        return f"Person(name='{self.name}', age={self.age})"


>>> john = Person('John Cleese', age=80)
>>> repr(john)
"Person(name='John Cleese', age=80)"
>>> print(john)
Person(John Cleese)
>>> john2 = eval(repr(john))
>>> type(john2)
<class 'repr_person.Person'>
>>> id(john)
4472330616
>>> id(john2)
4472331736
```

CUSTOM OBJECTS

```
class User:
    def __init__(self, username, password):
        self.username = username
        self.password = password

    def __str__(self):
        return self.username

    def __repr__(self):
        return f"User(username='{self.username}', password='{self.password}')"
```

```
>>> from user import User
>>> u = User('jcleese', 'ParrotIsNoMore')
>>> str(u)
'jcleese'
>>> str([u])
"[User(username='jcleese', password='ParrotIsNoMore')]"
```

NEXT UP...



How print() has changed in Python 3

TABLE OF CONTENTS

PYTHON 2 vs 3

- How print() is called is one of the more obvious changes between Python 2 and Python 3
- In Python 2, print was a statement and was called without brackets

```
# Python 2
>>> print 'hello world'
```

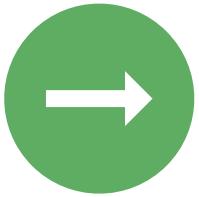
- print() is now a function
  - Replaces a lot of special syntax (trailing commas, >> redirects) with simple arguments to the function
  - Can now use advanced function mechanisms such as injection, composition and mocking like any other function

FUNCTION REFERENCE

- Functions are first class citizens in Python, they can be passed by reference like any other object

```
def download(url, log=print):
    log(f'Downloading {url}')
    # ...

def silence(*args):
    pass  # Do not print anything

download('/js/app.js', log=silence)

# ... or
download('/js/app.js', lambda msg: print('[INFO]', msg))
```

NEXT UP...



Pretty print: an alternative to print() for displaying objects

TABLE OF CONTENTS

pprint()

```
>>> from pprint import pprint
>>> data = {
...     'squares':[x**2 for x in range(10)]
... }
>>> pprint(data)
{'squares': [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]}
>>> data['tens'] = [x**10 for x in range(10)]
>>> print(data)
{'squares': [0, 1, 4, 9, 16, 25, 36, 49, 64, 81], 'tens': [0, 1, 1024, 59049, 1048576, 9765625, 60466176, 282475249, 1073741824, 3486784401]}
>>> pprint(data)
{'squares': [0, 1, 4, 9, 16, 25, 36, 49, 64, 81],
 'tens': [0,
          1,
          1024,
          59049,
          1048576,
          9765625,
          60466176,
          282475249,
          1073741824,
          3486784401]}
>>> pprint(data, width=20)
{'squares': [0,
             1,
             4,
             9,
             16,
             25,
             36,
             49,
             64,
             81],
 'tens': [0,
          1,
          1024,
          59049,
          1048576,
          9765625,
          60466176,
          282475249,
          1073741824,
          3486784401]}
```

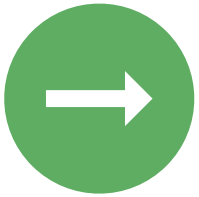
MORE pprint()

```
>>> pprint(data, indent=3, width=20)
{  'squares': [  0,
                 1,
                 4,
                 9,
                 16,
                 25,
                 36,
                 49,
                 64,
                 81],
   'tens': [  0,
              1,
              1024,
              59049,
              1048576,
              9765625,
              60466176,
              282475249,
              1073741824,
              3486784401]}
>>> pprint(data, width=40, compact=True)
{'squares': [0, 1, 4, 9, 16, 25, 36, 49,
             64, 81],
 'tens': [0, 1, 1024, 59049, 1048576,
          9765625, 60466176, 282475249,
          1073741824, 3486784401]}
```

MORE pprint()

```
>>> cities = {
...     'USA': {'Texas': {'Dallas':['Irving']}},
...     'CANADA': {'BC': {'Vancouver':['North Van']}},
... }
>>> pprint(cities)
{'CANADA': {'BC': {'Vancouver': ['North Van']}},
 'USA': {'Texas': {'Dallas': ['Irving']}}}
>>> pprint(cities, depth=3)
{'CANADA': {'BC': {'Vancouver': [...]}}, 'USA': {'Texas': {'Dallas': [...]}}}
>>> items = [1, 2, 3]
>>> items.append(items)
>>> print(items)
[1, 2, 3, [...]]
>>> pprint(items)
[1, 2, 3, <Recursion on list with id=4547300424>]
>>> id(items)
4547300424
>>> pprint('One', 'Two')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/pprint.py", line 53, in pprint
    printer.pprint(object)
  File "/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/pprint.py", line 139, in pprint
    self._format(object, self._stream, 0, 0, {}, 0)
  File "/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/pprint.py", line 176, in _format
    stream.write(rep)
AttributeError: 'str' object has no attribute 'write'
```

NEXT UP...



Printing in full color through the use of terminal escape sequences

TABLE OF CONTENTS

ANSI ESCAPE SEQUENCES

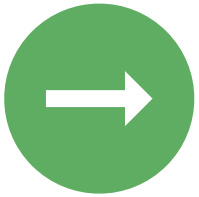
- Some terminals support the ability to send an escape code to control the formatting of text
- Most Unix & MacOS terminals support at least 16 colors, some up to true color
- Windows support is more recent, but you can achieve similar effects using a library: **colorama**
- The Wikipedia article on ANSI has a listing of all the escape codes:

  https://en.wikipedia.org/wiki/ANSI_escape_code

COLOR WITH ANSI

```
def esc(code):
    return f'\033[{code}m'

>>> print('this is ', esc('31'), 'really', esc(0), ' important', sep='')
this is really important
>>> print('this is ', esc('31;1'), 'really', esc(0), ' important', sep='')
this is really important
>>> print('this is ', esc('31;1;4'), 'really', esc(0), ' important', sep='')
this is really important
```

NEXT UP...



Using print() to do simple animations

TABLE OF CONTENTS

SIMPLE ANIMATION

- Recall two escape characters discussed in lesson #1:
  - **\r** -- Carriage return, returns the cursor to the beginning
  - **\b** -- Backspace
- By using these two together you can create simple flip-book style animations

SPINNING WHEEL

```
#!/usr/bin/env python
from time import sleep

# show the spinning animation 3 times
print('Everybody look busy  ', end='', flush=True)
for x in range(3):
    for frame in r'-\|/-\|/':
        # back up one character then print our next frame in the animation
        print('\b', frame, sep='', end='', flush=True)
        sleep(0.2)

# back up one character, print a space to erase the spinner, then a newline
# so that the prompt after the program exists isn't on the same line as our
# message
print('\b ')
```

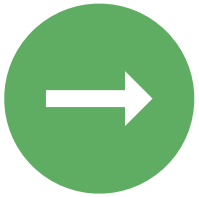
PROGRESS BAR

```
#!/usr/bin/env python
from time import sleep

# show the spinning animation 3 times
print('Everybody look busy  ', end='', flush=True)
for x in range(3):
    for frame in r'-\|/-\|/':
        # back up one character then print our next frame in the animation
        print('\b', frame, sep='', end='', flush=True)
        sleep(0.2)

# back up one character, print a space to erase the spinner, then a newline
# so that the prompt after the program exists isn't on the same line as our
# message
print('\b ')
```

OTHER SOURCES

- It is possible to create rich and complex interfaces just using the terminal
- “TUI” (Text User Interfaces) can be built with print(), but it is a lot of work
- Libraries to build complicated animations, games, or terminal based windowing systems:
  - **progressbar2**
  - **curses** or **ncurses**
  - **urwid**

NEXT UP...



How to write tests when your code uses print()

TABLE OF CONTENTS

TESTING

- When building non-trivial code you should have automated tests
  - Faster than testing by hand
  - Helps prevent regressive defects
- print() has a side-effect: displaying output, how do we ensure it works?
- Not testing print() itself, but that our method does what it is supposed to

```
def greet(name):
    print(f'Hello {name}')
```

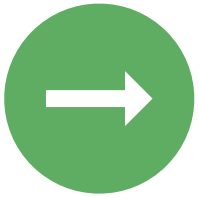
TESTING THROUGH INJECTION

```
#!/usr/bin/env python
from time import sleep

# show the spinning animation 3 times
print('Everybody look busy  ', end='', flush=True)
for x in range(3):
    for frame in r'-\|/-\|/':
        # back up one character then print our next frame in the animation
        print('\b', frame, sep='', end='', flush=True)
        sleep(0.2)

# back up one character, print a space to erase the spinner, then a newline
# so that the prompt after the program exists isn't on the same line as our
# message
print('\b ')
```

TESTING THROUGH MOCK

```
def greet(name):
    print('Hello ', name)


from unittest.mock import patch

@patch('builtins.print')
def test_greet(mock_print):
    # the actual test
    greet('John')
    mock_print.assert_called_with('Hello ', 'John')
    greet('Eric')
    mock_print.assert_called_with('Hello ', 'Eric')

    # showing what is in mock
    import sys
    sys.stdout.write(str( mock_print.call_args ) + '\n')
    sys.stdout.write(str( mock_print.call_args_list ) + '\n')
```

NEXT UP...



Using print() to debug and when to use logging instead

TABLE OF CONTENTS

DEBUGGING

- Often the hardest part of programming is squashing the bugs
- print() is a quick and easy way to help you hunt down those annoying errors in your code
- Sometimes called “caveman” debugging

DEBUGGING WITH PRINT

```
def movie_year(number):
    roman = { 1:'I', 5:'V', 10:'X', 50:'L', 100:'C', 500:'D', 1000:'M', }

    output = []
    divisor = 1000
    print(f'movie_year({number})')
    for digit in [int(x) for x in str(number)]:
        print(f'   digit={digit} divisor={divisor}')
        if 1 <= digit <= 3:
            # e.g. 3 -> 3*'I' -> 'III'
            output.append( roman[divisor] * digit )
        elif digit == 4:
            output.append( roman[divisor] +  roman[divisor * 5] )
        elif 5 <= digit <= 8:
            output.append( roman[divisor * 5] + roman[digit] * (divisor - 5) )
        elif digit == 9:
            output.append( roman[divisor] + roman[divisor * 10] )

        divisor = int(divisor / 10)
        print('      ', output)

    return ''.join(output)
```

LOGGING

- print() for debugging has some problems:
  - Not all software runs in a terminal (e.g. applications on servers or in devices)
  - May have to look for clues about a bug long after it occurs
  - Not thread safe!
- Logging sends output to a file
- There are tools for aggregating, searching and rotating logs

CALLING THE LOGGER

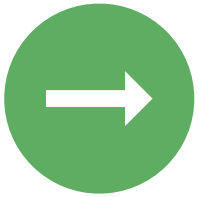
```
# counters.py

import logging, sys
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)
logger = logging.getLogger(__name__)

def count_lower_case(item):
    logger.info('count_lower_case(%s)', item)
    num = 0
    for letter in item:
        if 97 <= ord(letter) <= 122:
            logger.debug('  letter *%s* is lowercase', letter)
            num += 1

    logger.info('  returning %s', num)
    return num
```

CONFIGURING LOGGERS

- Python's logging module is very configurable

```
LOG_FORMAT_ESCAPED = '\033[1m%(funcName)s\033[0m: %(message)s'

root = logging.getLogger()
root.setLevel(log_level)

handler = logging.StreamHandler()
handler.setLevel(log_level)
handler.setFormatter(logging.Formatter(LOG_FORMAT_ESCAPED))

root.addHandler(handler)
```

NEXT UP...



Further information and useful libraries

TABLE OF CONTENTS

FURTHER STUDY

- Other courses:
  - Python 3's f-strings: An Improved String Formatting Syntax
    https://realpython.com/python-f-strings/
  - Logging in Python
    https://realpython.com/python-logging/
  - Adding Unit Tests
    https://realpython.com/lessons/adding-unit-tests/
  - Understanding the Python Mock Object Library
    https://realpython.com/python-mock-library/
  - Python Debugging with pdb
    https://realpython.com/python-debugging-pdb/

USEFUL LIBRARIES

- **ncurses** / **curses** -- terminal manipulation
- **urwid** -- TUI building toolkit
- **colored** -- ANSI colour printing
- **bullet** -- interactive command-line prompts
- **prompt-toolkit** -- interactive command-line application toolkit
- **questionnaire** -- tools for creating command-line questionnaires