Appendix D
A Summary of Python

As we said in the preface, we've assumed that you'd already been introduced to Python before you picked up this book. So the following amounts to a 'cheat sheet' for the Python features used here. Many Python topics aren't covered, such as class inheritance and decorators, since we don't use them. The appendix ends with three examples to show off Python in complete programs.

If you're looking for a Python tutorial, the official one at `https://docs.python.org/3/tutorial/` is excellent. We also recommend two online text books:

- *Think Python* by Allen B. Downey at `https://greenteapress.com/wp/think-python-2e/`
- *Automate the Boring Stuff with Python* by Al Sweigart at `https://automatetheboringstuff.com/`

D.1 Comments

```
# Single line comments start with a hash
'''
  Multiline strings can be written
  using three 's or "s.
'''
```

D.2 Variables

```
# A variable name is a.. zA..Z followed by a..zA..Z_0..9
```

```
age = 25        # integer; unlimited size
height = 1.65   # float; 64-bit

x, y = 25, 1e-9  # multiple assignment
x, y = y, x      # swap two values

isStudent = True  # (or False); a boolean
x = None    # None is a constant meaning 'undefined'
```

D.3 Operators

```
# Arithmetic: +, /, -, *
x = 5 / 2   # division; x = 2.5
x = 5 // 2  # integer division; x = 2
r = 5 % 2  # modulus; r = 1
s = 5 ** 2  # exponentiation; s = 25

x += 3  # same as x = x + 3
x -= 2  # same as x = x - 2

# Equality is ==
1 == 1  # => True
2 == 1  # => False
# Inequality
1 != 1  # => False
2 != 1  # => True

# Also: >, <, >=, <=

# Logical ops
g = True and False  # There's also: or
i = not True
```

D.4 Strings

```
"This is a string."
'This is also a string.'
name = "Alice"

"Hello " + "world!"  # => "Hello world!"
"Hello world!"[0]  # => 'H'

len("This is a string")  # => 16

# f-strings (f == formatting)
name = "Jane"
```

```
f"She said her name is {name}."
             # => "She said her name is Jane"
```

D.5 I/O

```
s = input("What is s? ")   # prompt for a string
x = int(input("x? "))   # read in an integer
x, y = map(float(input("x y? ").split()))
     # read in two floats separated by white space

name = "Alice"
print("Hello", name)   # Hello Alice and a newline
print("Hello", name, end = ' ')  # space instead of newline

age = 25
print("My name is", name, "age", age)
# My name is Alice age 25

print("Her age is " + str(age) + " years")
   # Her age is 25 years
print(f"My name is {name} and I am {age} years old.")
   # My name is Alice and I am 25 years old.

# f-strings for formatting numerical output
val = 12.3456789
print(f"val = {val:.3f}")  # prints 12.346
print(f"val = {val:9.3f}")  # prints ___12.346
```

D.6 Lists

```
fs = ["apple", "banana", "orange", "cherry"]
print(fs[0])  # apple
print(fs[1])  # banana
print(fs[-1]  # cherry; counts from right

fs[1] = "kiwi"
print(fs)     # ['apple', 'kiwi', 'cherry']

xs = [1]*10    # create a 10-element list, full of 1's

lst.append(item)   # add item to end of lst
lst.extend(lst2)   # add all of list lst2 to end
lst = lst + lst2   # concatenation

"banana" in fs    # True
```

```
fs.index("banana")  # 1
fs.index("durian")  # raises a ValueError exception

lst.insert(idx, val) # insert val at index
lst.remove(val)      # remove first item with value = val
x = lst.pop(idx)     # remove item at index and return its value

lst.sort()
xsSort = sorted(lst)    # create a sorted copy

lst.reverse()
revlst = reversed(lst)  # create a reversed copy

r = len(fs) # 4

lst = [1, 2, 3, 4]
min(lst)  # 1
max(lst)  # 4
sum(lst)  # 20
```

D.6.1 List Slices.

```
a = [0, 1, 2, 3, 4, 5]
a[1:]    # [1,2,3,4,5]
a[:5]    # [0,1,2,3,4]
a[:-2]   # [0,1,2,3]
a[1:3]   # [1,2]
a[1:-1]  # [1,2,3,4]

b=a[:]   # Shallow copy of a
```

D.6.2 2D Lists.

```
joeMarks = [97, 90, 92, 87]
jillMarks = [92, 79, 85, 88]
andrewMarks = [6, 13, 7, -1]
classMarks = [joeMarks, jillMarks, andrewMarks]
print(classMarks)

print(classMarks[2])     # row
# [6, 13, 7, -1]
print(classMarks[0][3])  # row column
# 87
```

D.6.3 List Comprehensions.

```
ns = [1, 2, 3, 4, 5]
squares = [n**2 for n in ns]
print(squares)  # [1, 4, 9, 16, 25]
```

```
[x for x in [3, 4, 5, 6, 7] if x > 5]  # => [6, 7]

xs = [x for x in [1,2,3,4,5,6,7,8,9] if x % 2 == 0]
# xs = [2,4,6,8]
```

D.7 Tuples

```
# Like lists but cannot be modified; they are immutable
tup = (1, 2, 3)
tup[0]      # => 1
tup[0] = 3  # Raises a TypeError

# You can use most of the list operations
len(tup)         # => 3
tup + (4, 5, 6)  # => (1, 2, 3, 4, 5, 6)
tup[:2]          # => (1, 2)
2 in tup         # => True
```

D.8 Dictionaries

```
# Dictionaries store key-value pairs
d =  {'Mary':'555-6789', 'Jenny':'867-5309',
      'John':'555-1234', 'Bob':'444-4321'}
print(d["John"])  # '555-1234'

d.keys()   # return ['Mary', 'Jenny', 'John', 'Bob']
d.values()
  # return ['555-6789', '867-5309', '555-1234', '444-4321']

for key in sorted(phoneNos.keys()):
  print(key, phoneNos[key])

v = d.pop('John')
# remove 'John':'555-1234', and v = '555-1234'
```

D.9 Sets

```
s1 = set {"keyl","key2"}
s2 = {1,9,3,0}
s3 = set()  # same as {}

nums1 = {1, 2, 3, 4}
nums2 = {2, 3, 5}
# set ops
```

```
nums1 & nums2    # intersection: {2, 3}
nums1 | nums2    # union: {1, 2, 3, 4, 5}
nums1 - nums2    # difference: {1, 4}
nums1 ^ nums2    # xor: {1, 4, 5}
2 in nums1     # True
10 in nums1    # False
```

D.10 If Statements

```
age = 5
# prints "age is smaller than 10"
if age > 10:
  print("age is totally bigger than 10.")
elif age < 10:
  print("age is smaller than 10.")
else:
  print("age is indeed 10.")

# Indentation is significant in Python!
# Convention is to use four spaces, not tabs.
# But we prefer to use two spaces.
```

D.11 Loops

D.11.1 For Loops.

```
for i in [0,1,2,3,4]): # iterate over lists
  print(i)

animals = ["dog", "cat", "mouse"]
for i, value in enumerate(animals):
    print(i, value)
```

D.11.2 Loops using range().

```
# range([start,] stop, [, step])

for i in range(5):   # [0,1,2,3,4]
  print(i)

for i in range(1,5):   # [1,2,3,4]
  print(i)

for i in range(0,10,2):   # [0, 2, 4, 6, 8]
  print(i)

for i in range(4,-1,-1):   # [4, 3, 2, 1, 0]
```

```
  print(i)

'''
  One of the most common beginner's mistakes
  is to forget that range() does not include
  the last element.
'''
```

D.11.3 while Loops.

```
x = 0
while x < 4:
  print(x)
  x += 1

while True:
  print("Hello", end = ' ')
```

D.11.4 Inside a Loop.

```
for i in [1, 2, 3, 4]:
  if i == 2:
    continue   # jump to start of next loop
  if i == 3:
    break      # end  loop
  print i      # prints => 1, 3
```

D.12 Functions

```
def adder(x, y, z=0):
  sum = x + y + z
  return sum

r = adder(1, 2)      # r = 3
r = adder(1, 2, 3)   # r = 6

# Return multiple values
def pows(x, y):
  return x**y, y**x

x = 2
y = 3
p1, p2 = pows(x, y)     # p1 = 8, p2 = 9
```

D.12.1 Function Names.

```
def add(x,y):
  return x + y
```

```
def subtract(x,y):
  return x - y

def calc(fn, x, y):     # function name argument
  return fn(x, y)       # call function

z = calc(add, 5, 10)      # z = 15
z = calc(subtract, 5, 10) # z = -5
```

D.12.2 Global Variables.

```
'''
  Global variables can be seen by every function,
  but can not be changed unless the function uses
  the global keyword.
'''

def tax(p, tax_rate):
  global taxCount
  total = p + (p * tax_rate)
  taxCount += 1
  return total

taxCount = 0
p = float(input("Enter a price: "))
p = tax(p, 0.06)
p = tax(p, 0.06)
p = tax(p, 0.06)
print(f"Price (after {taxCount} calls = {p}")
```

D.12.3 Lambda Functions.

```
f = lambda x: x**2
print(f(3))  # Output: 9

g = (lambda x, y: x**2 + y**2)
g(2, 1)  # => 5
```

D.12.4 map and filter.

```
def add10(x):
  return x + 10
list(map(add10, [1, 2, 3]))   # [11, 12, 13]

isOdd = lambda x: x%2 == 1
list(filter(isOdd, [3, 4, 5, 6, 7]))  # [3, 5, 7]
list(filter((lambda x: x%2 == 0), [3, 4, 5, 6, 7]))  # [4,6]
```

D.13 Exceptions

```
try:
  x = 1/0
except ZeroDivisionError:
  print("Cannot divide by zero")

try:
  database.update()
except Exception as e:
  print(e.msg)
  database.abort()
finally:  # always do this
  database.commit()
```

D.14 File I/O

```
# Read a file
with open("myfile.txt", "r") as f:
  contents = f.read()
print(contents)

# Write to a file
with open("myfile.txt", "w") as f:
  f.write("Hello, World!")
```

D.15 Modules

```
import math
print(math.sqrt(16))  # => 4.0

from math import ceil, floor
print(ceil(3.7))   # => 4
print(floor(3.7))  # => 3

# You can import all functions from a module
from math import *

# shorten module names
import math as m
math.sqrt(16) == m.sqrt(16)  # => True

'''
  Python modules are just ordinary Python files. You
  can write your own, and import them. The name of the
  module is the name of the file.
```

```
'''
from frange import *      # import everything from frange.py

'''
  Inside a module file, use __name__ check so test
  code is only run when the file is called directly
'''
if __name__ == '__main__':
  # test frange functions
```

D.16 Classes

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

  def greet(self):  # methods
    print("Hello, my name is " + self.name)

alice = Person("Alice", 25)
alice.greet()    # Hello, my name is Alice
alice.age = 26
```

D.17 Three Examples

These examples can be found at `http://coe.psu.ac.th/~ad/explore/code/D_Python/`.

D.17.1 `caesar.py`. caesar.py uses brute force on the encrypted text "nzohfu gur rarzl ng avtug" to crack its Caesar cipher (`https://en.wikipedia.org/wiki/Caesar_cipher`). The code illustrates the use of for-loops, if-statements, and f-strings in print().

```
codedMsg = 'nzohfu gur rarzl ng avtug'

for i in range(26):
  guess = ''
  for ch in codedMsg:
    if ch != ' ':
      # Shift the letter forward by i
      if ord(ch) + i <= 122:
        ch = chr(ord(ch) + i)
      else:  # Subtract 26 if we go beyond z
        ch = chr(ord(ch)+i-26)
    guess = guess + ch
  print(f"{chr(i+65):2s}: {guess}")
```

Listing D.1. Implementing the Caesar cipher

The output:

```
> python caesar.py
A : nzohfu gur rarzl ng avtug
B : oapigv hvs sbsam oh bwuvh
       : # lines not shown
M : zlatrg sgd dmdlx zs mhfgs
N : ambush the enemy at night
O : bncvti uif fofnz bu ojhiu
       : # lines not shown
Y : lxmfds esp pypxj le ytrse
Z : mynget ftq qzqyk mf zustf
```

The 'N' line reveals the plain text message.

D.17.2 `sinApprox.py`. sinApprox.py calculates the sine of an angle using its Taylor series approximation to N terms, and prints the result along with the math.sin() answer. The program illustrates the use of input conversion, while-loops, and the math module.

The Taylor series for sine:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

```
import math
N = 25

degs = float(input("degrees? "))
x = math.radians(degs)

i = 1
sum = x
sign = 1
while i < N:
  sign = -sign
  i += 2
  term = sign*x**i/math.factorial(i)
  sum += term

print(f"     sin(x) = {sum} ({N} terms)")
print("math.sin(x) =", math.sin(x))
```

Listing D.2. Approximating a sine

Typical output:

```
> python sinApprox.py
degrees? 30
```

```
     sin(x) = 0.49999999999999994 (25 terms)
math.sin(x) = 0.49999999999999994

> python sinApprox.py
degrees? 45
     sin(x) = 0.7071067811865475 (25 terms)
math.sin(x) = 0.7071067811865476
```

D.17.3 `romans.py`. romans.py is a small module containing functions for converting integers into Roman numerals (`https://en.wikipedia.org/wiki/Roman_numerals`), and vice versa. The module contains a test-rig after a `__name__` test.

```
# 1000 <->"M", 900 <->"CM", 500 <->"D"...
decimalDens = [1000,900,500,400, 100, 90, 50, 40, 10, 9, 5,4,1]
romanDens   = ["M","CM","D","CD","C","XC","L","XL","X","IX","V","IV",
    "I"]

def num2Roman(dec):
  # sanity checks
  if dec <= 0:
    raise ValueError("It must be a positive")
  elif dec >= 4000:
    raise ValueError("It must be lower than MMMM(4000)")
  return toRoman(dec, "", decimalDens, romanDens)


def toRoman(num, s, decs, romans):
  if decs != []:
    if num < decs[0]:
      # deal with the rest of the denomination
      return toRoman(num, s, decs[1:], romans[1:])
    else:
      # reduce this denomation till num < desc[0]
      return toRoman(num-decs[0], s+romans[0], decs, romans)
  else:
    return s


def roman2Num(str):
  str = str.upper()
  res = 0
  i = 0
  while (i < len(str)):
    s1 = toDecimal(str[i])
    if (i+1 < len(str)):
      s2 = toDecimal(str[i+1])
      if (s1 >= s2):
        res += s1
        i += 1
      else:  # s1 < s2
```

```
        res += (s2 - s1)
        i += 2
    else:
      res += s1
      i += 1
  return res

def toDecimal(ch):
  # Roman symbol to decimal
  try:
    idx = romanDens.index(ch)
    return decimalDens[idx]
  except ValueError:
    print(ch, "not recognized; using 1")
    return 1



if __name__ == "__main__":
  rs = []
  for n in [12, 44, 57, 111, 1910, 2024]:
    r = num2Roman(n)
    rs.append(r)
    print(n, r)
  print("---------------")
  for rom in rs:
    print(rom, roman2Num(rom))
```

Listing D.3. Integers to/from Roman numerals

When the program is run from the command line, the test-rig is called:

```
> python romans.py
12 XII
44 XLIV
57 LVII
111 CXI
1910 MCMX
2024 MMXXIV
---------------
XII 12
XLIV 44
LVII 57
CXI 111
MCMX 1910
MMXXIV 2024
```

If the module is imported (in this case, to the Python REPL), then the test-rig is not executed, but the functions are available:

```
>>> from romans import *
>>> num2Roman(1999)
```

```
'MCMXCIX'
>>> roman2Num("MCMXCIX")
1999
```