UC San Diego

CSE6R NOTES

Summarized course notes by Emyl Safin bin Mohd Zaky

Instructor: Moshiri Niema

SP23

Table of Contents

The Basics

1.1 Variables

We pass data to the computer through the use of **variables**. Variables can be assigned a value which will have a **basic data type** and can be plugged into expressions like in math. Basic Data Types:

int - Holds any **whole number**.

```
example_one = 9
```

float - Holds any **decimal**.

```
example_two = 3.845
```

bool - Holds either **True** or **False**.

```
example_three = False
```

str - Holds **one or more characters**, represented with double or single quotes. (Note that it is common to call variables of type str as strings)

```
example_four = "This is a string."
```

Note that in Python we can assign multiple variables in one line as shown in the example below.

```
foo, bar, baz = True, "Gauss", -2.3411
```

There is a fifth special basic data type called **NoneType**.

NoneType - When there is no data to be stored in a variable, assigning the value **None**.

```
example_none_type = None
```

1.2 Strings

1.2.1 Zero-based numbering

Consider the string below:

```
lilith = "Daughter of Hatred"
```

In Python, data is **zero-indexed**. Meaning that the **initial element** of a sequence is assigned the **position(index) 0** instead of the position 1.
Hence in the line above the first element or index 0 is D, the third element or index 2 is u, and so forth.

1.2.2 Indexing

We can verify the fact above through the use of the **indexing operator**, which allows us to access elements at specific indexes. For example:

```
lilith = "Daughter of Hatred"
first = lilith[0] # Indexing Operator
print(first)
```

Output

D

Substrings

We call **sequential characters** in a string a **substring**. The indexing operator can be used to extract substrings. In general we do this by:

```
foobar = ??? # Arbitrary string
substring = foobar[start : end + 1]
```

Here start represents the starting position and end is the ending position of the substring.
Note that the +1 is due to the **end index** being **exclusive** in Python.
We can see this work more specifically in the example below:

```
diablo = "Lord of Terror"
substring = diablo[8 : 14]
print(substring)
```

Output

Terror

String length

We can call the **len function** to find the **length** of a string. For example:

```
area = "Sarn"
length_of_area = len(area)
print(length_of_area)
```

Output

4

It can also be called directly on the string as such:

```
length = len("Highgate")
```

Say we wished to obtain the following about a substring:

1. Prefix of string up to index N
2. Suffix of string starting at index N

We could obtain them as such:

```
foobar = 'crucible'
prefix = foobar[:N+1]
suffix = foobar[N:]
```

Note that I have used shorthand notation above.
For the prefix when I exclude the start value Python will assume it is 0, similarly for the suffix when I exclude the end value Python will assume it is the last index +1.

Negative indices

Negative indices are valid in Python.
A **negative index** $-n$ represents the **n th character from the end**. For example:

```
ring = "9282321"
mid = ring[-4]
print(mid)
```

Output

2

1.2.3 + Operator

We can use the **+ operator** to join multiple strings. For example:

```
front = "La"
mid = "T"
back = "eX"
front_mid_back = front + mid + back
print(front_mid_back)
```

Output

LaTeX

Strings can only be concatenated with other strings.
To concatenate a string with another data type we would get its **string representation** by calling the function **str()**. For example:

```
a = "Pay"
b = 2
c = "Win"
a_b_c = a + str(b) + c
print(a_b_c)
```

Output

Pay2Win

1.2.4 Representing long strings

Say we had a variable that we wanted to assign to a very long string. We could write it out in one line but that would be less legible.
Instead we could use **backslash** "\" to **join multiple strings declared on different lines**. For example:

```
long_string = "touch grass is not an insult towards gamers, "\
              "rather it is advice for them. "\
              "When participating in intense periods of gaming, "\
              "the human hand has a tendency to get sweaty."\
              "The sweat causes the hand to become slick, "\
              "and it becomes more difficult to retain a grip "\
              "on the gamer's gaming mouse."
print(long_string)
```

This would give the same result as declaring the entire string as one.
Not important but this behavior is caused by Python's use of implicit concatenation and how Python handles physical and logical lines.

String with multiple lines.

We can store a **multiline string** into a sigle variable by the use of the **\n** "newlines" each time we wish to have a line break. For example:

```
archive_details = "Number 28.\n\nJackson Pollock, American\n1912-1956\n\nEnamel on
    canvas."
```

Alternatively we could use three quotation marks to improve legibility as shown below:

```
archive_details = """Number 28.

Jackson Pollock, American
1912-1956

Enamel on Canvas."""
print(archive_details)
```

Output

```
Number 28.

Jackson Pollock, American
1912-1956

Enamel on Canvas.
```

1.3 Printing

To get your program to output text we call the function **print()**. This is called printing to **standard output**.
We have already seen examples of this function in some of the sample code above. To use this function, we just specify whatever you want displayed in the parentheses.

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

Output

```
Hello World!
```

As seen in earlier examples using print() on a variable will print the value of that variable. It is also important to note that **indexing and + operators can be used with print()**. For example:

```
truth = 'redrum '
word = 'equals '
print(truth + word +truth[-2::-1])
```

Output

```
redrum equals murder
```

Note that the -1 above denotes the step of the indexing operator.

```
string = ??? #Arbitrary string
string[ start: end: step] #Returns substring from index start to index end, at step
     size step.
```

By default the print() function adds a \ n at the end of whatever we are printing.
We can change this behavior and determine the character(s) added at the end of print() by:

```
print('Wraeclast' , end=' & ')
print('Oriath', end='.')
```

Output

```
Wraeclast & Oriath.
```

1.4 Operators

Addition	+	
Subtraction	-	
Multiplication	*	
Division	/	
Integer division	//	$3//2 = 1$
Remainder (modulo)	%	
Exponentiation	**	
Negation	-	

1.4.1 Operator Precedence

From highest to lowest:

()	Parentheses
**	Exponentiation
$-$	Negation
$*, /, //, \%$	Multiplication , division , remainder
$+, -$	Addition , Subtraction
$=, +=, -=,$ etc.	Assignment

1.5 Comments

There are two ways we can comment on a piece of code in Python. Single line comments and multi line comments.

Single line comments

We use the character ' # ' to make single line comments. For example:

```
#Declare assignments
plug = 'CIA'
bird = 'Saint'
soldier = 'Manboy'

print(plug + '+' + bird) #Print key assignments
```

For single line comments we place the comment **above** if we're describing a **block of code** and **on the same line to the right** if we're describing **a line of code**.

Multi line comments

We use quote marks to denote multi line comments. A multi line comment starts with three consecutive quote marks and ends with three consecutive quote marks. For example:

```
'''
Program to convert kilometers to miles.
'''

kilometers = float(input("Distance in kilometers:"))

conv_fac = 0.621371

miles = kilometers * conv_fac
print('%0.2f kilometers is equal to %0.2f miles' %(kilometers , miles))
```

Conditionals

2.1 Boolean algebra