Python Reference Sheet

The LiFi-project

Prof. Dr. Nicolas Meseth

> **Disclaimer**
>
> This is a reference sheet for students who start their programming journey in the context of the LiFi project. This sheet covers many of the Python commands we use in this project, but certainly not all of them. It is important to note that this guide is not intended to be a comprehensive introduction to Python, but rather a specific reference sheet for the LiFi project.

Overview

Commands

We utilize some commands over and over in our examples and exercises. The most important commands are listed in this section.

Printing to the Console

The `print` function outputs information to the console. We can use a so-called *format string* by prefixing the string with `f`. This allows us to use placeholders that are replaced with their actual values when the program runs:

```
year = 2023
print(f"Welcome to the year { year }")

# Output: Welcome to the year 2023
```

In contrast, if we skip the `f`, the result is very different and probably not what we wanted:

```
year = 2023
print("Welcome to the year { year }")

# Output: Welcome to the year { year }
```

I recommend you always use the `f`-prefix in your programs.

Importing modules

We frequently use external code in our programs. This is good because it means we don't have to reinvent the wheel over and over again. We utilize the `import` command to tell our program that we want to use an external module.

```
import math
print(f"The constant PI = { math.PI }")

# Output: The constant PI = 3.14
```

There is an alternative syntax for importing specific elements from a module:

```
from math import PI
print(f"The constant P= { PI }")

# Output: The constant PI = 3.14
```

The difference is that we do not load the entire module, which can save memory, and that we can skip the module's name when referencing the imported element. In this latter case, we can reference the constant with `PI` instead of `math.PI`.

Variables

We can store intermediate results in variables to access them later on in our code:

```
a_whole_number = 42
a_decimal_numer = 3.14
a_string = "LiFi"
a_boolean = True
```

The above example initialize variables and store primitive values of different data types. We can also store more complex things on variables, such as lists, dictionaries, functions, or references to our Tinkerforge devices. We can really use them to store anything.

Control Structures (`if` and `else`)

Solving problems requires decision-making. In programming, we make decisions using so-called control structures. The most important one is the `if`-statement.

```
a_number = input("Please enter a positive number: ")

if a_number < 0:
  print(f"Oops, the number { a_number } is negative.")
else:
  print(f"Well done! You entered a positive number.")
```

The above example checks the given condition and does the one or the other, depending on the outcome. Note that the condition expression (here: `a_number < 0`) must evaluate to either `True` or `False`. We call these kinds of expression a *boolean expression*.

We can also define multiple cases with `elif`:

```
age = input("Please enter your age: ")

if age < 0:
  print(f"Oops, your age cannot be negative.")
elif age < 6:
  print(f"Sorry, this content is not suitable for children under the age of 6.")
elif age <= 12:
  print(f"You will be able to see content suitable for ages 6 to 12.")
elif age <= 18:
  print(f"You will be able to see content suitable for ages 12 to 18.")
else:
  print(f"You are old enough to access all content.")
```

Loops (`while` and `for`)

Problem-solving frequently involves doing the same thing multiple times. Depending on how we can decide how many times we must do the same thing, we can use either a `while` or a `for` loop.

```
while command != "exit":
  command = input("Please enter a command!")
```

When we have list of items and we need to do something for every item, we can use a `for`-loop:

```
numbers = [1, 3, 5, 7, 11]

for n in numbers:
  print(f"Currently looking at the number { n }")

# Output:
# Currently looking at the number 1
# Currently looking at the number 3
# Currently looking at the number 5
# Currently looking at the number 7
# Currently looking at the number 11
```

We can use the `for`-loop for counting, too:

```
for i in range(5):
  print(f"Counter i = { i }")

# Output:
# Counter i = 0
# Counter i = 1
# Counter i = 2
# Counter i = 3
# Counter i = 4
```

Functions

We define functions with the `def` keyword followed by a name of our choice. A function can have one or more parameters:

```
# Define a new function with two parameters x and y
def multiply(x, y):
  product = x * y
  return product

# Calling the new function
result = multiply(4, 5)
print(result)
```

```
# Output: 20
```

Expressions and Operators

- Comparison operators: `==`, `>`, `<`, `>=`, `<=`, `!=`
- Arithmetic operators: `+`, `-`, `*`, `/`, `%`
- Logical operators: `and`, `or`, `not`

Collections

- Lists
- Dictionaries
- Sets

Working with Strings

Determine Number of Characters

The `len` function lets us determine how long a string is:

```
a_string = "Welcome to the LiFi project"
print(f"The string has { len(a_string) } characters")

# Output: The string has 27 characters
```

Splitting Strings

The `split` functions can separate a string into multiple chunks:

```
a_string = "Welcome to the LiFi project"

# This creates a list with individual words
chunks = a_string.split(" ")

print(chunks)
```

```
# Output: ['Welcome', 'to', 'the', 'LiFi', 'project']
```

Working with Tinkerforge Devices

Connecting to the Devices

For the LiFi project, the following boilerplate code established a connection to your devices and stores references to each of them on a separate variable. Make sure you replaced the UID in your personal version of `constants.py`.

```
import constants

from tinkerforge.ip_connection import IPConnection
from tinkerforge.bricklet_rgb_led_v2 import BrickletRGBLEDV2
from tinkerforge.bricklet_rotary_encoder_v2 import BrickletRotaryEncoderV2
from tinkerforge.bricklet_color_v2 import BrickletColorV2
from tinkerforge.bricklet_oled_128x64_v2 import BrickletOLED128x64V2

ipcon = IPConnection() # Create IP connection
ipcon.connect(constants.HOST, constants.PORT) # Connect to brickd

# Create device instances
led = BrickletRGBLEDV2(constants.UID_RGB_LED, ipcon)
rotary = BrickletRotaryEncoderV2(constants.UID_ROTARY_ENCODER, ipcon)
oled = BrickletOLED128x64V2(constants.UID_OLED_DISPLAY, ipcon)
color = BrickletColorV2(constants.UID_COLOR_SENSOR, ipcon)
```

You are now ready to use the devices in your code.

RGB LED

Coming soon.

Rotary Encoder

Coming soon.

OLED 128x64 Display

Coming soon.

Color Sensor

Coming soon.