

piRover Builds with K2

piRover – User Blink

Rev 1.0

Overview:

In this activity you learn to use a variable to hold a value. You will set a value for the variable using the Python input function. You'll then use an IF statement to test the value of the variable. Based on the state of the variable you will change the state of the piRover's LEDs.

The requirements for the solution are listed below.

- The user will see a welcome message indicating that this is the User Inputs and Decision activity.
- The user will be prompted for which LED to blink. Valid values will be red, green, and blue. The user will be notified if an invalid value is entered.
- The user will be prompted for the blink speed, entering either fast or slow. The user will be notified if an invalid value is entered.
- The solution will blink the LEDs using the color and relative speed specified.

Prerequisites:

Prior to beginning the instruction provided in this lesson you must have completed the following:

1. piRover Blink

Performance Outcomes:

1. Use the print() function to inform the user
2. Use the input() function to prompt the user for a value
3. Create a variable and set the value using the input() function.
4. Test the value of a variable using the IF/ELIF/ELSE structure and control which segment of code is executed.
5. Use variables as arguments in functions such as sleep()
6. Review GPIO code from Blink activity

Resources:

1. [Variables in Python](#)
2. [If statements in Python](#)
3. [Input function in Python](#)

Materials:

1. piRover

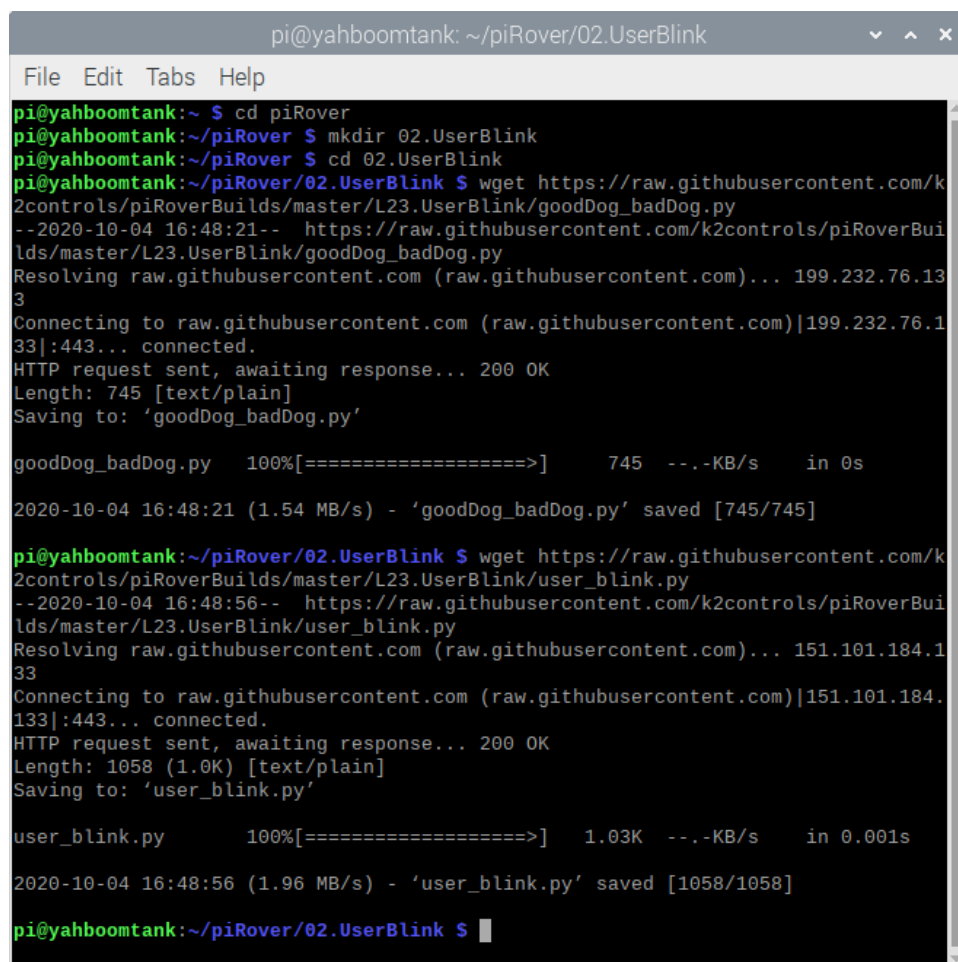
piRover Builds with K2

Part 1 – Set Up

1. Preview the overview of variables on [Variables in Python](#) resource.
2. Preview the concepts of decisions using if statement using this [if statements in Python](#) resource.
3. Preview the input() function using the [Input function in Python](#) resource.
4. Prepare your workspace for this activity.
 - a. Connect to your piRover using VNC.
 - b. Access your piRover folder
 - c. Create a 02.UserBlink directory
 - d. Change to the 02.UserBlink directory
 - e. Download the starter files for the activity. Copy and paste the wget instructions below.

wget https://raw.githubusercontent.com/k2controls/piRoverBuilds/master/L23.UserBlink/goodDog_badDog.py

wget https://raw.githubusercontent.com/k2controls/piRoverBuilds/master/L23.UserBlink/user_blink.py



```
pi@yahboomtank: ~/piRover/02.UserBlink
File Edit Tabs Help

pi@yahboomtank:~ $ cd piRover
pi@yahboomtank:~/piRover $ mkdir 02.UserBlink
pi@yahboomtank:~/piRover $ cd 02.UserBlink
pi@yahboomtank:~/piRover/02.UserBlink $ wget https://raw.githubusercontent.com/k2controls/piRoverBuilds/master/L23.UserBlink/goodDog_badDog.py
--2020-10-04 16:48:21-- https://raw.githubusercontent.com/k2controls/piRoverBuilds/master/L23.UserBlink/goodDog_badDog.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 199.232.76.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|199.232.76.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 745 [text/plain]
Saving to: 'goodDog_badDog.py'

goodDog_badDog.py  100%[=====>]      745  --.-KB/s    in 0s
2020-10-04 16:48:21 (1.54 MB/s) - 'goodDog_badDog.py' saved [745/745]

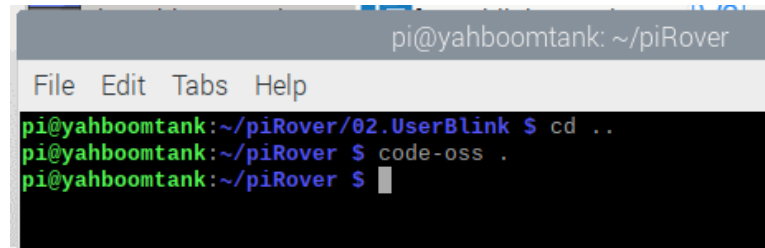
pi@yahboomtank:~/piRover/02.UserBlink $ wget https://raw.githubusercontent.com/k2controls/piRoverBuilds/master/L23.UserBlink/user_blink.py
--2020-10-04 16:48:56-- https://raw.githubusercontent.com/k2controls/piRoverBuilds/master/L23.UserBlink/user_blink.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.184.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.184.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1058 (1.0K) [text/plain]
Saving to: 'user_blink.py'

user_blink.py      100%[=====>]    1.03K  --.-KB/s    in 0.001s
2020-10-04 16:48:56 (1.96 MB/s) - 'user_blink.py' saved [1058/1058]

pi@yahboomtank:~/piRover/02.UserBlink $
```

piRover Builds with K2

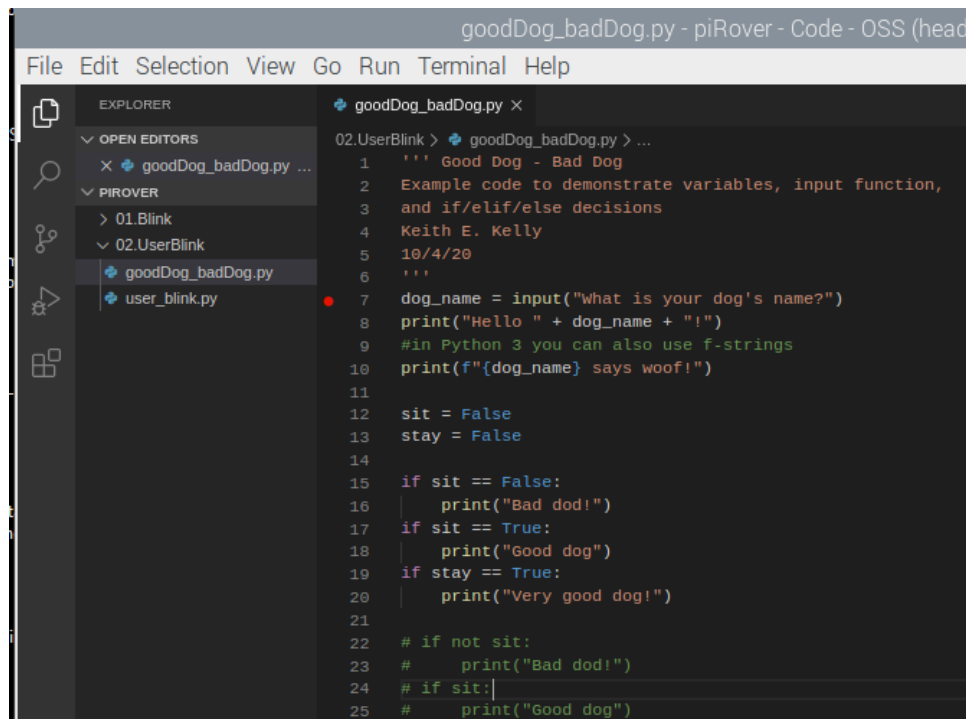
5. Move back to the piRover parent directory using the `cd ..` command and launch VS Code.



```
pi@yahboomtank: ~/piRover
File Edit Tabs Help
pi@yahboomtank:~/piRover/02.UserBlink $ cd ..
pi@yahboomtank:~/piRover $ code-oss .
pi@yahboomtank:~/piRover $
```

Part 2 – Introducing Variables, input(), and decisions

6. Open the `goodDog_badDog.py` file in the VS Code editor. Set a breakpoint at line 7. Press F5 and follow along with the instructor as the concepts and use of variables, input, and decisions are explained.



```
goodDog_badDog.py - piRover - Code - OSS (head
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
x goodDog_badDog.py ...
PIROVER
  01.Blink
  02.UserBlink
    goodDog_badDog.py
    user_blink.py
goodDog_badDog.py x
02.UserBlink > goodDog_badDog.py > ...
1  ''' Good Dog - Bad Dog
2  Example code to demonstrate variables, input function,
3  and if/elif/else decisions
4  Keith E. Kelly
5  10/4/20
6  '''
7  dog_name = input("What is your dog's name?")
8  print("Hello " + dog_name + "!")
9  #in Python 3 you can also use f-strings
10 print(f"{dog_name} says woof!")
11
12 sit = False
13 stay = False
14
15 if sit == False:
16     print("Bad dod!")
17 if sit == True:
18     print("Good dog")
19 if stay == True:
20     print("Very good dog!")
21
22 # if not sit:
23 #     print("Bad dod!")
24 # if sit:
25 #     print("Good dog")
```

7. Here are a list of take-aways from `goodDog_badDog` example.
 - a. `dog_name` is a variable. It is a container for the name entered by the user.
 - b. `dog_name` is lower case and uses and underscore to separate words. This is a standard in the Python language.
 - c. `dog_name` holds a series of characters returned from the keyboard via the `input()` function. This is variable is know as having a “string” data type (string of characters).

piRover Builds with K2

- d. The `input()` function returns the string of characters that a user types until she or he presses the Enter key.
- e. Function parameters are values that are provided to the function and located inside the parentheses that follows. The `input` function accepts a string parameter prompting the user for what value to enter, in this case “What is your dog’s name?”
- f. The quotation marks are used around the string parameter on line 7 because this is literally what you want displayed. Later you might include a variable here.
- g. The `print()` function is used to display text in the terminal window. In line 8 a string value is provided.
- h. String values can be combined or “concatenated” using the `+` operator.
- i. In line 8, two literal strings are combined with the string variable to form the output message to the user.
- j. Starting with Python 3.6, a more efficient method for combining strings and variables was introduced. The f-string is used on line 10. Note the `f` just prior to the literal string value and the curly braces used around the variable name. Also note how the color coding in VS Code assists.
- k. In addition to string, variable can also hold numbers and Boolean values. Boolean values are either `True` or `False`.
- l. Lines 12 and 13 create two new Boolean variables named `sit` and `stay`. You can change the values from `False` to `True` to change the behavior of the code that follows.
- m. In lines 15 through 20, the values of `sit` and `stay` are tested using the `if` statement. Note the use of the colon (`:`) after the `if` statement.
- n. The `if` statement must contain a condition that evaluates to a Boolean value. In line 15, the value of the `sit` variable is compared to `False`. Note the use of the double equal signs.
- o. If the condition evaluates to `True` then the lines of code that are indented after the `if` are executed.
- p. Compare line 12 to line 15 to understand the use of a single equal sign to the use of a double equal sign.

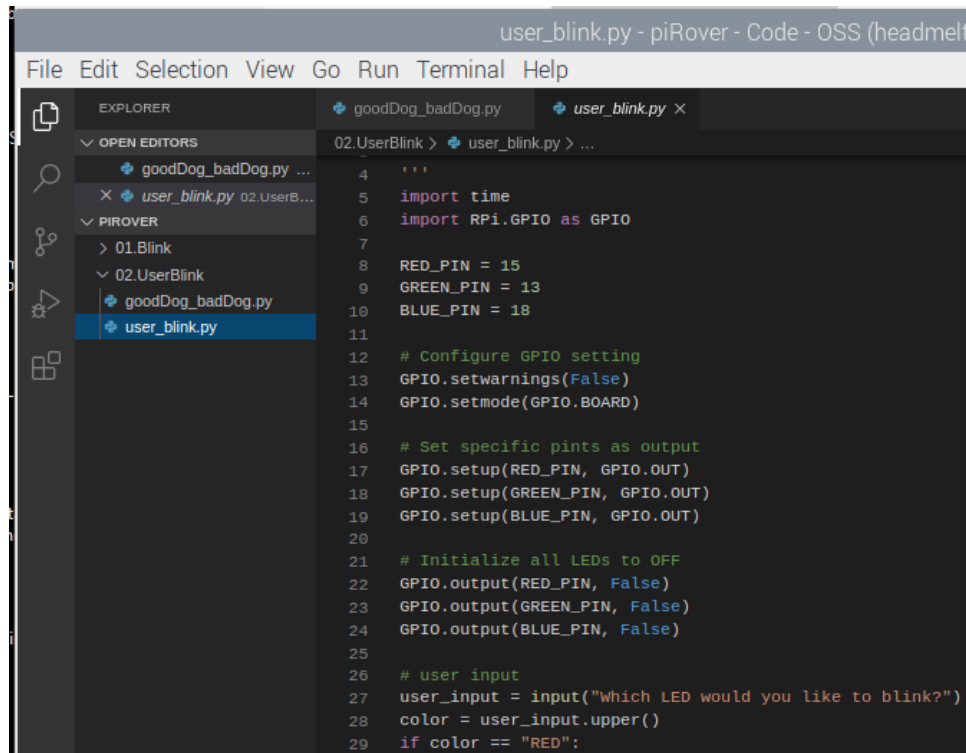
piRover Builds with K2

- q. In line 12 the right side of the equal sign is being evaluated and “assigned” to the sit variable of the left side. This single equal sign is known as the assignment operator.
 - r. In line 15 a question is being answered or tested...is the value of sit equal to the value of False – yes or no? The double equal sign is known as a comparison operator and is checking the two values to see if they are equal. The result is a Boolean value (True or False).
 - s. The if statement requires that the condition that follows evaluates to a Boolean value. In line 22, the code is shortened by recognizing the sit is already a Boolean. Note the use of the not operator her to test for a False value.
 - t. Two logical operators that are used often are “and” and “or”. In line 26 the “and” operator is used to test both the sit and stay variables. Do you understand line 28 or are you confused?
 - u. In lines 31 through 36, the if/elif/else structure is shown. Here a series of values is checked, and the corresponding output is produced. The block always starts with “if” and you can include as many “elif” conditions as necessary.
 - v. Note that the if and elif statements require a Boolean condition. The else is known as a fall-through. This code block runs if none of the prior conditions evaluate to True. Thus, there is no condition code following else, just the colon.
8. On your own...
- a. Create a variable to hold the dog’s age. Use the assignment operator (=) to set a value.
 - b. Create a new if statement testing the dog’s age. If the dog is over 10 years old, then print “Old dog...no new tricks!” Use the greater than (>) comparison operator in your test condition.
 - c. Can you extend this to check if the dog is a puppy? Dogs under 2 are still puppies. Create the required output.

piRover Builds with K2

Part 3 – User Blink

9. Open the user_blink.py in the VS Code editor.



```
user_blink.py - piRover - Code - OSS (headmelt)
File Edit Selection View Go Run Terminal Help

EXPLORER
OPEN EDITORS
  goodDog_badDog.py ...
  X user_blink.py 02.UserB...
PIROVER
  01.Blink
  02.UserBlink
    goodDog_badDog.py
    user_blink.py

4 '''
5 import time
6 import RPi.GPIO as GPIO
7
8 RED_PIN = 15
9 GREEN_PIN = 13
10 BLUE_PIN = 18
11
12 # Configure GPIO setting
13 GPIO.setwarnings(False)
14 GPIO.setmode(GPIO.BOARD)
15
16 # Set specific pints as output
17 GPIO.setup(RED_PIN, GPIO.OUT)
18 GPIO.setup(GREEN_PIN, GPIO.OUT)
19 GPIO.setup(BLUE_PIN, GPIO.OUT)
20
21 # Initialize all LEDs to OFF
22 GPIO.output(RED_PIN, False)
23 GPIO.output(GREEN_PIN, False)
24 GPIO.output(BLUE_PIN, False)
25
26 # user input
27 user_input = input("Which LED would you like to blink?")
28 color = user_input.upper()
29 if color == "RED":
```

10. Run the user_blink.py code. Does it function?
11. Note the pin definitions on lines 8, 9, and 10. These are variables, but the capitalized name indicates that they are “constants”, the value will never change during program.
12. Review the GPIO provided on lines 13 through 24. Do you recognize this code from the earlier Blink activity? Note how the pin names are used rather than the literal values of 13, 15, and 18. What if a pin value was changed by the manufacturer? How easy is this code to modify?
13. Could you create the initial GPIO code on lines 13 through 24 on your own?
14. Review the input code on lines 27 and 28. A new function is introduced – upper(). Why is this include? What is the value of adding this to the code?
15. The solution functions, but let’s refactor or improve the structure. Can you find duplicate code that can be eliminated by introducing a new variable?

piRover Builds with K2

16. Enter the revised code shown below. A new pin variable is defined on line 28

```
23 GPIO.output(GREEN_PIN, False)
24 GPIO.output(BLUE_PIN, False)
25
26 # Create a variable for the LED pin
27 # Set its default value to RED
28 pin = RED_PIN
29
30 # ask the user which LED to blink
31 user_input = input("Which LED would you like to blink?")
32 color = user_input.upper()
33 if color == "RED":
34     pin = RED_PIN
35 elif color == "GREEN":
36     pin = GREEN_PIN
37 elif color == "BLUE":
38     pin = BLUE_PIN
39 else:
40     print("Sorry, that is not a valid pin.")
41
42 while True:
43     GPIO.output(pin, True)
44     time.sleep(1)
45     GPIO.output(pin, False)
46     time.sleep(1)
47
```

17. Run the revised solution. Does it function? Is this better code?
18. On your own...
- Follow the pattern show for the pin selection to create the ability for the user to specify the blink rate – either fast or slow.
 - Define a delay variable. If the delay is set to 1 then the blink rate is slow, but if the delay is set to .3 then the blink rate is fast.
 - Test your code for all combinations of user inputs.
 - What happens if the user enters an invalid blink speed?
19. Once you completed the solution, copy your 02.UserBlink folder up to your cloud service as a backup (OneDrive, Google Drive, or DropBox)

Assessment:

Submit your final **user_blink.py** file to Moodle along with other files in this week's zip file.