KETE HS24

Introduction to Python Programming

TUTORIAL 2

repl.it

Created by Daniel Benninger

Inhaltsverzeichnis

Tutorial II

Intermediate Python Programming

1 – String Manipulation.

Learning Goals/Objectives

Be able to read, comprehend, trace, adapt and create Python code that:

- Uses **len** to count the number of characters in a string
- Uses *get char* to get a single character from a string
- Uses **substring** to get a sequence of characters from a string
- Uses *change case* to convert caps to lower case and vice versa

repl.it

Theory - String Manipulation

A **string**?

- **Strings** are a **data type** used by Python.
- All data stored in string variables is treated as **text**, even if it is numeric characters.
- String data is surrounded by **“quotation marks”**

```
“This is a string”
```

```
variable = “This is a string being stored in a variable”
```

repl.it

A **function**?

We have already learned how to create our own functions, but Python also has lots built in.

- They are pre-set sections of code that perform common tasks.
- They have a name and often use **parameters.**
- Functions **return** a value to the program.

→ *We are going to learn about the functions that we can use with strings.*

```
functionName(parameter1,parameter2)
```

repl.it

len()

Returns the number of characters in a string

Human **len()**

len() counts the number of characters in a string and returns the number to the program.

What would the following function calls return?

len(“orange)
len(“Computing.”)
len(“Hello World!”)

repl.it

len() into a variable

When the value is returned, we need to store it somewhere.

We use a variable for this.

```
stringLength = len("Hello World!")
```

1 - Give the variable a sensible name.

2 - Use the len function with the string as the parameter.

repl.it

Getting len() of input

1 - Get input just like we have done previously

```
word = input("Enter a word)
wordLength = len(word)


print(wordLength)
```

3 - The number of characters is returned into the *wordLength* variable.

2 - Use the variable with the string in it as a parameter.

4 - Output the variable with the number of characters stored in it

repl.it

Getting len() of input - More efficient code

1 - Get input just like we have done previously

```
word = input("Enter a word)

print(len(word))
```

2 - Use the len function as a parameter of the print function.

repl.it

len with selection


```
word = input("Enter a word)
wordLength = len(word)

if wordLength > 50:
    print("Wow! That's a long word!"
```

1 - Use the *wordLength* variable as part of the condition.

repl.it

len with selection - more efficient code


```
word = input("Enter a word)

if len(word) > 50:
    print("Wow! That's a long word!")
```

1 - Use the *len()* function as part of the condition.

repl.it

Len - Independent Task

Write a program that:

- Asks the user to input an 8 letter word.
- Stores the input in a variable.
- Calculates the length of the word input.
- If the word is more than 8 characters, output ‘Too long’
- If the word is less than 8 characters, output ‘Too short’
- If the word is 8 characters, output ‘Perfect, thank you!’

repl.it

Get Char - String Slicing

Get Char?

- Get Char lets us get a single character from a string
- Python treats strings like lists, each character is given its own index number, starting at 0

0	1	2	3	4	5
P	y	t	h	o	n

repl.it

Using **Slices** to get a character

- We use code called a **slice** to get a substring in Python.
- Slices treat strings like **lists.** Each character is given an index number starting with 0

1. Assign the text to a variable.

```
phrase = "Hello World!"
letter = phrase[4]
print(letter)
```

2. **Slices out character 4 of the string in the *phrase* variable.**

3. Output the variable containing the slice

Get Char - Make

Write a program that:

- Asks the user to input a word and stores it in a suitably named variable
- Asks the user to input a number and stores it in a suitably named variable.
- If the number entered is larger than the length of the word input then output an error message.
- Else output the character from the word at the position input as part of a sentence.

Eg for inputs ‘Jimi’ and ‘2’ the program outputs ‘The letter **m** is at position **2** in your name’.
For inputs ‘Jimi’ and ‘6’ the program outputs ‘The number you entered is too large’.

repl.it

Substrings - String Slicing

A **substring**?

- A **substring** is part of a string.
- It is **some text from a string** exactly as it appears in the string

```
"This is a string"
```

```
This
This is
is a
is is a st
```

repl.it

A substring

Which of the examples below is a substring of:

“Computer Science”

A) `comp`
B) `uter Sci`
C) `ter Science.`
D) `CompSci`

Using **Slices** to get substrings

- We use code called a **slice** to get a substring in Python.
- Slices treat strings like **lists.** Each character is given an index number starting with 0

1. Assign the text to a variable.

```
phrase = "Hello World!"
subPhrase = phrase[2:5]
print(subPhrase)
```

2. **Slices out characters 2 to 5 of the string in the *phrase* variable.**
Character 5 is **NOT INCLUDED.**

3. Output the variable containing the slice

repl.it

Using **Slices** to get substrings - more efficient code


```
phrase = "Hello World!"
print(phrase[2:5])
```

repl.it

Task - Substrings - Make

Write a program that:

- Asks the user to input a phrase and stores it in a suitably named variable
- Asks the user to input a number between 0 and the length of the phrase and stores it in a suitably named variable.
- Asks the user to input a second number between the first number and the length of the phrase and stores it in a suitably named variable.
- Gets and outputs the substring of characters between the two numbers entered

Eg for inputs ‘I love Computing’ and numbers ‘3’ and ‘7’ the output would be ‘ove’

repl.it

https://repl.it/@MrAColley/17-Substrings-Predict-Run

Change Case

Case?


```
"THIS IS UPPER CASE"

"this is lower case"
```

repl.it

Changing To Uppercase

```
“Hello World!”.upper()

word.upper()
```

1. Put the string or string variable first.

2. Put a . then `upper()`

repl.it

Changing To Lower

```
“Hello World!”.lower()

word.lower()
```

1. Put the string or string variable first.

2. Put a `.` then `lower()`

repl.it

Change case - more efficient code

```
print(“Hello World”.upper())

print(word.upper())
```

repl.it

Change Case - Using In Conditions

```
name = “dAvE”
nameUpper = name.upper()

If nameUpper == “DAVE”:
   print(“Hi Dave”)
```

repl.it

Change Case - Using In Conditions

```
name = “dAvE”

if name.upper() == “DAVE”:
```

Tutorial II

Intermediate Python Programming

2 – Numbers.

Learning Goals/Objectives

Be able to read, comprehend, trace, adapt and create Python code that uses:

- Data *types & casting* - understanding the different data formats that variables & lists use, and how to convert between them.
- **Random** - how to generate random numbers and use them in programs
- **Modulo** - how to calculate the remainder of an integer division and why this is useful

repl.it

Data Types

What is a data type?

- A **data type** is a setting for a **variable**. It tells the variable what sort of data it will store.
- At the moment, we have only used two data types, the **string** and the **int.**
  - What type of data does a **string** store?
  - What type of data does an **int** store?

Data Types in Python

String - Text

Int - Whole numbers

Float - Numbers with decimals.

(also called *Real* in other coding languages)

Type - How To Code

The **type()** function returns the data type of some data or a variable

Put the data in the brackets as a parameter.

```
type(5.4)


num1 = 5
print(type(num1))
```

OR put the variable containing the data in the brackets as a parameter.

repl.it

Programming – **Casting** (Changing One Data Type To Another)

```
int(data/variable)
float(data/variable)
str(data/variable)
```

repl.it

Programming – Casting Int

```
x = int(1)      # x will be 1
y = int(2.8)    # y will be 2
z = int("3")    # z will be 3
```

Programming – Casting Float

```
x  = float(1)          # x will be 1.0
y  = float(2.8)        # y will be 2.8
z  = float("3")        # z will be 3.0
w = float("4.2")       # w will be 4.2
```

repl.it

Programming – Casting String

```
x = str("s1")
y = str(2)
z = str(3.0)
```

```
# x will be 's1’
# y will be ‘2’
# z will be '3.0'
```

repl.it

Casting - How To Code

Casting lets us convert one data type into another.

1. Put the name of the data type you want to convert **to.**

2. Put the data OR the variable in brackets as the parameter.

```
int(data/variable)
float(data/variable)
str(data/variable)
```

repl.it

Task - Predict & Run


```
1   # Task
2
3   # Add comments to the code to predict what the code does and what the output will be.
4
5   data1 = 2.9
6   data2 = "Hello World!"
7   data3 = 6
8
9   print(type(data1))
10  print(type(data3))
11  print(type(data2))
12
13  data1 = data1 + 0.1
14
15  print(type(data1))
16
17  data3 = float(data3)
18  data1 = int(data1)
19
20  print(type(data1))
21  print(type(data3))
```

repl.it

Task - Investigate & Modify


```
2
3   num1 = int(input("Enter a number"))
4   num2 = float(input("Enter another number"))
5
6   total = num1 * num2
7
8   print(total)
9
10    # What does the 'int' before 'input' make the program do?
11
12    # What does the 'float' before 'input' make the program do?
13
14    # What data type will the output be?
15
16
17  #Task - Modify
18
19    #Adapt the code to:
20
21    # Get both inputs as floats
22    # Convert the total to an int before it is output.
23
```

https://repl.it/@MrAColley/2202-Type-and-Cast-Investigate-and-Modify

Task - Make


```
1   # Task - Make
2
3   #Write a program to calculate the area of a rectangle.
4
5   # Get input for height & width as floats.
6
7   #Multiply the height & width to calculate the area.
8
9   #Output the area as part of a sentence.
10
11  #Extra challenge - create this as a function with a suitable name
    that takes fixed height & width as its parameters.
12
13  # Extra extra challenge - create this as a function with a suitable
    name that takes user input as its parameters.
14
```

repl.it

Random Numbers

Random Numbers in Python

Python has lots of pre-written features & functions that we can use.

Often, these features are grouped together and called **libraries.**

To use a library we have to **import** it (you only have to do this once).

It is common practice to put all of your imports at the **top** of your code.

→ The library we are going to use for random numbers is called **random**

repl.it

Random- How To Code

The **randint(x,y)** function takes 2 parameters.
x is the lowest random number that can be picked. y is the largest.
The program will pick a number between these two limits.

1. Import the library (you only have to do this **once**)

```
import random
```

2. Type the name of the library, a . and the name of the function you want to use.

```
random.randint(1,20)
```

3. Put your upper and lower limits in the brackets as parameters.

repl.it

Task - Predict & Run


```
1   # Task Predict & Run
2
3   #Add comments to explain what the code does and what the output will be.
4
5   import random
6
7   print(random.randint(1,5))
8
9   num1 = random.randint(1,10)
10
11  print("Your number was " + str(num1))
12
```

repl.it

Task - Investigate & Modify


```
1   # Task - Investigate
2
3   # Answer the questions below the code.
4   import random
5
6   print("Welcome to the dice simulator!")
7
8   num1 = random.randint(1,6)
9
10  print("You rolled a " + str(num1))
11
12  # What is the term for the (1,6) values used by the randint function?
13
14  # Why are the numbers in brackets not (0,6)
15
16  # What would the effect be if the last two lines of code swapped places?
17
18  # Task - Modify
19
20  #Adapt the code so that
21
22  #It generates a second number by rolling the dice again.
23
24  #It adds the two dice rolls together.
25
26  #It outputs the total of the two dice rolls
```

repl.it

Task - Make


```
1   # Task - Make
2
3   #Write a program that:
4
5   #Gets user input of two numbers.
6
7   # Extra challenge - Build in a check for the input, if the second number is lower than or
    the same as the first number then output an error message. Else continue to the next steps.
8
9   #Generates a random number between the two numbers input.
10
11  # Outputs the random number generated.
12
```

repl.it

Modulus

Modulus in Python

3 MOD 1 = 3 remainder 0 → so the value returned would be **0**

5 MOD 2 = 2 remainder 1 → so the value returned would be **1**

14 MOD 4 = 3 remainder 2 → so the value returned would be **2**

What Value Will Be Returned By….

7 MOD 6

29 MOD 4

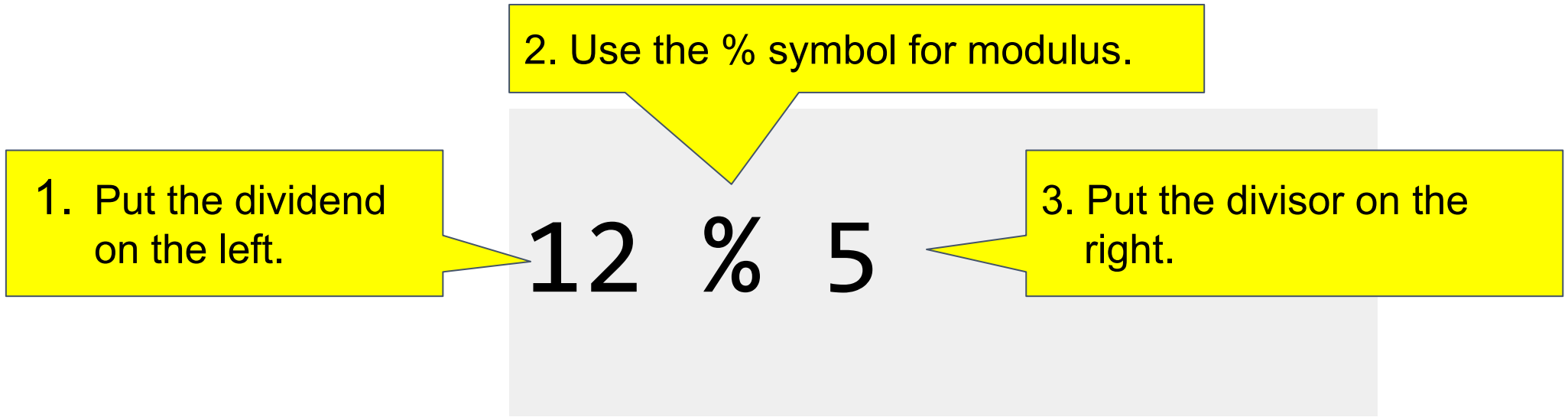
15 MOD 5

9 MOD 7

35 MOD 4

Modulus - How To Code

The **modulus** operator returns the **remainder** of integer division.



repl.it

Modulus With Variables

The **modulus** operator returns the **remainder** of integer division.

```
num1 = 12
num2 = 5
remainder = num1 % num2
```

Task - Predict & Run


```
1   # Task - Predict & Run
2
3     # Add comments to the code to explain what it does and what the output
      will be.
4
5     # Run the code to test your predictions.
6
7   8 % 3
8   9 % 3
9   14 % 5
10
11  remainder = 10 % 4
12  print(remainder)
13
14  num1 = 16
15  num2 = 4
16
17  remainder = num1 % num2
18  print("The remainder is " + remainder)
19
```

repl.it

Task - Investigate


```
# Task - Investigate

# Answer the questions about the code below. Type your answers as comments.

num1 = int(input("Enter a number"))

print ("I will now calculate if your number is in the two times table.")

remainder = num1 % 2

if remainder == 0:
  print("Your number is in the two times table")
else:
  print("Your number is not in the two times table")

  # What is the purpose of the code?

  # What symbol is used for modulus?

  # What would happen if the input to the program was 17?

  # What would happen if the input to the program was 98?

  # What is the condition in the code?

  # Why can the code use 'else' instead of 'elif' and still work correctly?
```

repl.it

Task - Modify


```
# Task - Modify

  # Complete the code below so that:

  # It gets user input into the 'name variable'
  # It uses selection to output suitable messages depending on whether
  the name has an odd or even number of characters
  # Remove the need for the 'nameLength' and 'nameRemainder' variables by
  using len and modulus in the condition for the selection.

name = "Dave"

nameLength = len(name)
nameRemainder = nameLength % 2

if
```

repl.it

Task - Make


```
1   # Task - Make - The Love Calculator
2
3   # Write a program that:
4
5   # Gets two users to input their names.
6   # Calulates the number of characters in each name and adds them together.
7   # Calculates the modulus of the total characters in both names divided by
    3.
8   # If the modulus is 0, output a message saying that the couple are very
    compatible.
9   # If the modulus is 1, output a message saying that the couple are might
    have a chance together.
10  # If the modulus is 2, output a message saying that the couple aren't
    compatible.
```

Tutorial II

Intermediate Python Programming

3 – Selection.

repl.it

Learning Goals/Objectives

Be able to read, comprehend, trace, adapt and create Python code using selection that:

- Uses **Boolean operators** with multiple conditions.
- Checks if a number is inside or outside a range.
- Nests selection statements inside each other.

repl.it

More than one Boolean Operator

Boolean Operators

and - Checks at least two conditions. Returns true if **all** conditions are true

```
if hair == ‘blonde’ and eyes != ‘blue’ and hasPet == true:
    Stand_up
else:
    Sit_down
```

repl.it

Boolean Operators

`or` - Checks at least two conditions. Returns true if **at least one** condition is true

```
if hair == ‘blonde’ or eyes != ‘blue’ or hasPet == true:
    Stand_up
else:
    Sit_down
```

Boolean Operators

not - Inverts the value from the condition. Returns false if the condition is true and true if the condition is false.

```
if not (hair == ‘brown’):
     Stand_up
else:
     Sit_down
```

Multiple Booleans - How To Code

Use **and** or **or** between the conditions.

1. Set your data, either by assignment or input.

```
num1 = 23


if num1 < 50 and num1 < 100:
```

2. Put the Boolean operator between the conditions in your code

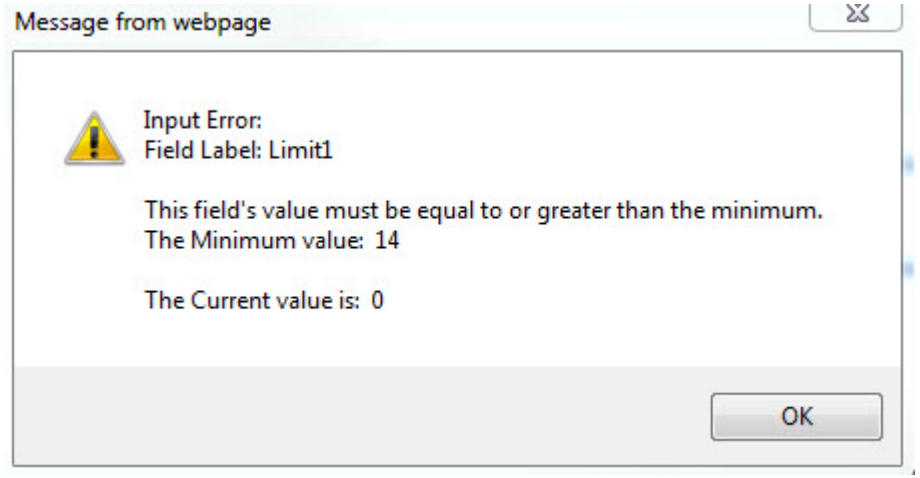
Numbers in a Range (Validation)

Validation

Checking whether data entered is **reasonable, sensible** and **allowable.**

Sometimes you only want to allow the user to input certain numbers.

We can use selection and Boolean conditions to produce error messages if they don't.



repl.it

Validation **inside** a range - how to code

Using selection will allow you to create success/error messages. It **won't** make the user try again - we're building up to that.

1. Get the input.

```
num1 = input("Enter a number between 1 and 10")

if num1 >= 1 and num1 <= 10:
    print("Number in range")
else:
    print("Number not in range")
```

2. Use **>=** the lowest allowable number **and <=** the highest allowable number.

3. Success message if it is in the range.
Else error message.

repl.it

Validation **outside** a range - how to code

Using selection will allow you to create success/error messages. It **won't** make the user try again - we're building up to that.

1. Get the input.

```
num1 = input("Enter a number between 1 and 10")

if num1 < 1 or num1 > 10:
    print("Number not in range")
else:
    print("Number in range")
```

2. Use < the lowest allowable number **or** > the highest allowable number.

3. Error message if it is in the range.
Else success message.

repl.it

Nesting

Nesting

Nesting is putting programming structures inside each other.

For example, an **if inside an if**.

```
num1 = 53
if num1 > 10:
        print(“More than 10”)

        if num1 > 30:
                print(“ and more than 30”)
        else:
                print(“ but not more than 30”)
```

Nesting Selection - How To Code


```
num1 = 53
if num1 > 10:
        print("More than 10")

        if num1 > 30:
                print(" and more than 30")
        else:
                print(" but not more than 30")
```

1. Create your first selection statement as normal.

2. Nest the next statement inside the fist by using 'Tab' to indent it.

3. This statement will only run if the condition in the first 'if' is true.

TAB

repl.it

Tutorial II

Intermediate Python Programming

4 – Iteration.

Learning Goals/Objectives

Be able to read, comprehend, trace, adapt and create Python code using selection that:

- Uses **conditional iteration** (while loops)
- **Validates** user input
- **Repeats whilst** the user does not input the desired data.

repl.it

Validation With **Conditional** Iteration

Validation

Checking whether data entered is **reasonable, sensible** and **allowable.**

Sometimes you only want to allow the user to input certain numbers.

We have used selection and Boolean conditions to produce error messages if they don't.

Now we are going to use iteration (loops) to keep getting input until the data is allowable.

repl.it

Validation - The Algorithm

1. Get the input & store in a variable.
2. Start the loop, set the condition to repeat whilst the input is not the same as the desired data.
3. Output an error message.
4. Get the input again, store in the same variable as before.
5. End the loop
6. Output a success message

Validation with text - how to code

1. Get the input.

2. Start a while loop with conditions that check for **un-desirable** input. The loop should keep running while the input is NOT what you want.

```
password = "pa55w0rD"
userPW = input("Enter your password")

while userPW != password:
    userPW = input("Password incorrect, try again")

print("Password accepted")
```

3. Get the input AGAIN INSIDE THE LOOP.

4. Success message once the loop has ended. It will only display this once the loop has stopped.

Validation with a range - how to code

Using iteration will allow you to make the user try again if the data entered is not allowable.

1. Get the input.

```
num1 = input("Enter a number between 1 and 10")


while num1 < 1 or num1 > 10:
      Num1 = input("Number not in range, try
   again please")

print("Thank you")
```

2. Start a while loop with conditions that check for **undesirable** input. The loop should keep running while the input is NOT what you want.

3. Success message once the loop has ended. It will only display this once the loop has stopped.

Task - Predict & Run


```
1   # Task - Predict Run
2
3   # Add comments to the code to explain how it will work.
4   # Run the code to check your predictions.
5
6
7   #Example 1
8
9   correctUserName = "Dave"
10
11  userName = input.lower(("Please enter your username"))
12
13  while userName != correctUserName:
14      userName = input("Username incorrect, please try again.")
15
16  print("Username accepted.")
17
18  # Example 2
19
20  num1 = input("Enter a number between 1 and 10")
21
22  while num1 < 1 or num1 > 10:
23      num1 = input("Invalid number, please try again")
24
25  print("Input accepted.")
```

repl.it

Task - Investigate


```
1   # Task - Investigate 1
2
3   # Answer the questions about the code
4
5   num1 = input("Enter a number between 50 and 100")
6
7   while num1 < 50 or num1 > 100:
8
9     if num1 < 50:
10     print("Invalid number, too small, try again.")
11
12    num1 = input("Enter a number between 50 and 100")
13
14
15  # Where is the iteration in the code?
16
17  # How many conditions are there in the code?  What are they?
18
19  # Where is the nesting in the code?
20
21  # Why are lines 10 and 12 indented?
22
23  # Why is line 10 indented twice?
24
25  # What would be the impact of swapping the 'or' on line 7 for an 'and'?
26
```

repl.it

Task - Modify


```
28  # Task - Modify
29
30  # Copy the code and adapt it so that:
31
32  # It asks the user to enter a the number of the month they were born (eg input 1 for
    January.)
33
34  # It validates the input against a sensible range and gives the user a chance to try
    again.
35
36  # It outputs a suitable error message if the user inputs a number that is too large.
37
38  # It outputs a suitable error message if the user inputs a number that is too small.
39
```

repl.it

Task - Make 1


```
1   # Task - Make 1
2
3   # Write a program that:
4
5   # Asks the user to choose a new password and input it twice.
6   # Validates the inputs and loops until they match
7   # Outputs a success message when they do match.
8
```

repl.it

Task - Make 2


```
1   # Task - Make 2
2
3   # Write a program that:
4
5   # Asks the user to choose a new password and input it twice.
6   # Validates each input so that the length has to be at least 8 characters.
7   # Validates the inputs to make ssure that they match.
8   # Outputs a success message when they do match.
9
```

repl.it

Tutorial II

Intermediate Python Programming

5 – Lists.

Resources created by Andy Colley (@MrAColley)

Learning Goals/Objectives

Be able to read, comprehend, trace, adapt and create Python code that:

- Uses lists to **store data**
- **Outputs** a range of items from a list
- **Searches** a list to find an item

repl.it

Output a Range of Items

Variable or List?

Variable - stores **one** piece of data with an identifier.

```
player1 = Mary
player2 = Sean
player3 = Atif
```

List - stores **more than one** piece of data with the same identifier.

```
players = [“Mary”, “Sean”, “Atif”]
```

List Range - How To Code

We can output a range from a list by customising the `print` command.

```
fruits = ["apple", "banana", "grapes", "strawberry", "orange"]

print(fruits[2:4])
```

This will start at index 2 (grapes).

This will end at index 3 (strawberry). The end number in the brackets is NOT included.

Use the list name in the print command.

List Range - Variations

```
fruits = [“apple”, “banana”, “grapes”, “strawberry”, “orange”]

print(fruits[:4])


print(fruits[2:])
```

Outputs from the beginning up to index 3 (strawberry).
The end number in the brackets is **NOT** included.

Outputs from index 2 (grapes) to the end of the list.

repl.it

Task - Predict & Run


```
1   # Task Predict & RuntimeError
2
3   # Add comments to the code to predict how it will work.
4   # Run the code to check your predictions
5
6   rockStars = ["John","Paul","George","Ringo","Freddie","Brian","John","Roger"]
7
8   print(rockStars[3:6])
9
10  print(rockStars[:5])
11
12  print(rockStars[4:])
```

repl.it

Task - Investigate


```
1   # Task Investigate
2
3   # Answer the questions about the code below
4
5   rockStars = ["John","Paul","George","Ringo","Freddie","Brian","John","Roger"]
6
7   listLength = len(rockStars)
8
9   print(rockStars[3:listLength])
10
11  # What does the len() function do?
12
13  # If 'Axl' was appended to the list, what would be the effect on the listLength variable?
14
15  # What would be the effect on the output?
16
```

repl.it

Task - Modify


```
18  # Task - Modify
19
20  # Copy the code and adapt it so that:
21
22  # It asks the user to input a number between 0 and the last index of the list (hint - you'll need to
    subtract 1 from the length of the list).
23
24  # It validates the input so that users can't enter a number smaller than 0 or bigger than index of
    the last item in the list.
25
26  # It asks the user to input a second number between the first number input + 1 and the last index of
    the list (hint - you'll need to subtract 1 from the length of the list).
27
28  # It validates the input so that users have to input a number bigger than the first input and less
    than or equal to  than index of the last item in the list.
29
30  # It outputs the items in the list between the two numbers input.
31
```

repl.it

Task - Make

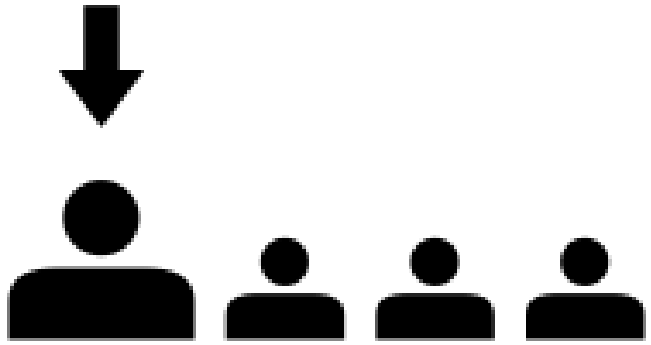

```
1   # Task - Make
2
3   # Write a program that:
4
5   # Stores 10 names in a list
6
7   # Asks the user to input 1 to choose output range, 2 to choose output from a point to the end
    of the list or 3 to choose output from the beginning of the list to a point.
8
9   # Validates the input and does not continue until it is suitable.
10
11  # Depending on the option chosen asks the user to input relevant start/end points.
12
13  # Outputs the relevant items from the list.
14
15  # Extra challenge - code the three different outputs as separate functions and call them when
    needed.
16
```

repl.it

Search A List

Linear

‘Resembling a line’.



Linear Search

Searching each item of data one after the other, starting with the first.

(Move along the line)

repl.it

0 1 2 3 4

Pets[]

repl.it

Search A List - How To Code - Method 1

Using a **while** loop

1. Initialise a **counter** variable to 0 to keep track of how many times the loop has run.

```
counter = 0
found   = False
```

2. Initialise a **found** variable to false. We will change this to true when/if we find our item in the list.

repl.it

Search A List - How To Code - Method 1

Using a **while** loop

```
counter = 0
found  = False

while counter < len(list):
```

3. Start a while loop. Set the condition to be while the counter variable is less than the length of the list.

repl.it

Search A List - How To Code - Method 1

Using a **while** loop

```
counter = 0
found  = False

while counter < len(list):
      if list[counter] == itemLookingFor:
            found = True

      counter +=1
```

4. Use selection. Use **counter** as the index of the item in the list that is being examined. If it is the same as the item we're looking for then set **found** to True.

5. **OUTSIDE** the selection but **INSIDE** the loop, increment counter by 1.

repl.it

Search A List - How To Code - Method 1

```
counter = 0
found  = False

while counter < len(list):
      if list[counter] == itemLookingFor:
            found = True
      counter +=1

if found == True:
      print(itemLookingFor + " has been found in the list")
else:
      print(itemLookingFor + " is not in the list")
```

4. **AFTER** the loop,use selection to display a message about whether the item was found or not.

Search A List - How To Code - Method 1


```
list = [“pencil”, “ruler”, “pen”, “eraser”, “calculator”]
itemLookingFor = “pen”
counter = 0
found  = False

while counter < len(list):
     if list[counter] == itemLookingFo
          found = True
counter +=1

if found == True:
     print(itemLookingFor + “ has been found in the list”)
else:
     print(itemLookingFor + “ is not in the list”)
```

itemLookingFor	counter	found	list[counter]
pen	0	False	

Search A List - How To Code - Method 1


```
list = [“pencil”, “ruler”, “pen”, “eraser”, “calculator”]
itemLookingFor = “pen”
counter = 0
found  = False

while counter < len(list):
     if list[counter] == itemLookingFo
          found = True
counter +=1

if found == True:
     print(itemLookingFor + “ has been found in the list”)
else:
     print(itemLookingFor + “ is not in the list”)
```

itemLookingFor	counter	found	list[counter]
pen	0	False	
pen	0	False	pencil

Search A List - How To Code - Method 1


```
list = [“pencil”, “ruler”, “pen”, “eraser”, “calculator”]
itemLookingFor = “pen”
counter = 0
found  = False

while counter < len(list):
     if list[counter] == itemLookingFo
          found = True
counter +=1

if found == True:
     print(itemLookingFor + “ has been found in the list”)
else:
     print(itemLookingFor + “ is not in the list”)
```

itemLookingFor	counter	found	list[counter]
pen	0	False	
pen	0	False	pencil
pen	1	False	ruler

Search A List - How To Code - Method 1


```
list = [“pencil”, “ruler”, “pen”, “eraser”, “calculator”]
itemLookingFor = “pen”
counter = 0
found  = False

while counter < len(list):
     if list[counter] == itemLookingFo
          found = True
counter +=1

if found == True:
     print(itemLookingFor + “ has been found in the list”)
else:
     print(itemLookingFor + “ is not in the list”)
```

itemLookingFor	counter	found	list[counter]
pen	0	False	
pen	0	False	pencil
pen	1	False	ruler
pen	2	True	pen

Search A List - How To Code - Method 1


```
list = [“pencil”, “ruler”, “pen”, “eraser”, “calculator”]
itemLookingFor = “pen”
counter = 0
found  = False

while counter < len(list):
     if list[counter] == itemLookingFo
          found = True
counter +=1

if found == True:
     print(itemLookingFor + “ has been found in the list”)
else:
     print(itemLookingFor + “ is not in the list”)
```

itemLookingFor	counter	found	list[counter]
pen	0	False	
pen	0	False	pencil
pen	1	False	ruler
pen	2	True	pen
pen	3	True	eraser

Search A List - How To Code - Method 1


```
list = [“pencil”, “ruler”, “pen”, “eraser”, “calculator”]
itemLookingFor = “pen”
counter = 0
found  = False

while counter < len(list):
     if list[counter] == itemLookingFo
          found = True
counter +=1

if found == True:
     print(itemLookingFor + “ has been found in the list”)
else:
     print(itemLookingFor + “ is not in the list”)
```

itemLookingFor	counter	found	list[counter]
pen	0	False	
pen	0	False	pencil
pen	1	False	ruler
pen	2	True	pen
pen	3	True	eraser
pen	4	True	calculator

Search A List - How To Code - Method 1


```
list = [“pencil”, “ruler”, “pen”, “eraser”, “calculator”]
itemLookingFor = “pen”
counter = 0
found  = False

while counter < len(list):
     if list[counter] == itemLookingFo
          found = True
counter +=1

if found == True:
     print(itemLookingFor + “ has been found in the list”)
else:
     print(itemLookingFor + “ is not in the list”)
```

itemLookingFor	counter	found	list[counter]
pen	0	False	
pen	0	False	pencil
pen	1	False	ruler
pen	2	True	pen
pen	3	True	eraser
pen	4	True	calculator
	5		

Search A List - How To Code - Method 2


```
list = [“pencil”, “ruler”, “pen”, “eraser”, “calculator”]
itemLookingFor = “pen”
counter = 0
found  = False

while counter < len(list):
     if list[counter] == itemLookingFor and found == False:
          found = True
counter +=1
```

itemLookingFor	counter	found	list[counter]
pen	0	False	
pen	0	False	pencil
pen	1	False	ruler
pen	2	True	pen
	3		

Search A List - How To Code - Method 3


```
list = [“pencil”, “ruler”, “pen”, “eraser”, “calculator”]
itemLookingFor = “pen”


if itemLookingFor in list:
     print(itemLookingFor + “ has been found in the list”)
else:
     print(itemLookingFor + “ is not in the list”)
```

Task - Predict & Run


```
# Task - Predict & Run

# Add comments to the code to predict what it will do when run.
# Run the code to test your predictions

######### Example 1 ######################

rockStars = ["John","Paul","George","Ringo","Freddie","Brian","John","Roger"]

counter = 0
found = False

while counter < len(rockStars):

  if rockStars[counter] == "George":
    found = True

  counter +=1

if found == True:
  print("George is in the list")
else:
  print("George is not in the list")
```

repl.it

Task - Predict & Run


```
25  ######## Example 2 ######################
26
27  if "George" in rockStars:
28    print("George is in the list")
29  else:
30    print("George is not in the list")
31
32  ######## Example 3 ######################
33
34  counter = 0
35  found = False
36
37  while counter < len(rockStars) and found == False:
38
39    if rockStars[counter] == "George":
40      found = True
41
42    counter +=1
43
44  if found == True:
45    print("George is in the list")
46  else:
47    print("George is not in the list")
```

repl.

Task - Investigate


```
1  ################ Task - Investigate
2
3  # Answer the questions about the code below
4
5  rockStars = ["John","Paul","George","Ringo","Freddie","Brian","John","Roger"]
6
7  counter = 0
8  found = False
9
10 while counter < len(rockStars):
11
12   if rockStars[counter] == "George":
13     found = True
14
15   counter +=1
16
17 if found == True:
18   print("George is in the list")
19 else:
20   print("George is not in the list")
21
22 # What is the purpose of the counter variable?
23
24 # What is the purpose of the found variable?
25
26 # What does the line counter +=1 do?
27
28 # If line 12 was changed to if rockStars[1] what effect would it have on the output and why?
29
30 # Why is the selection on line 17 not indented?
```

repl.it

Task - Modify


```
33  ############## Task - Modify
34
35  # Copy the code from above and adapt it so that
36
37  # It outputs the list to the user.
38  # The user has to input the item to be found.
39  # It ignores case on the input
40  # The program ends the loop when it has found the item
41
42
43  ############## Task - Modify 2
44
45  # Copy the code from above and adapt it so that it uses the python if...in instead of a while loop.
46
```

repl.it

Task - Make


```
1   # Task - Make
2
3   # Write a program that:
4
5   # Initialises a list of the top 10 selling artists of all time (do some
    research and find out who they are)
6
7   # Asks the user to guess an artist who could be in the list.
8
9   # Ignores case for the comparisons below.
10
11  # If the artist input is in the list, output a suitable message.
12
13  # If the input is not in the list, output a suitable message.
14
```

repl.it

Tutorial II

Intermediate Python Programming

6 – File Handling.

Resources created by Andy Colley (@MrAColley)

Learning Goals/Objectives

Be able to read, comprehend, trace, adapt and create Python code that:

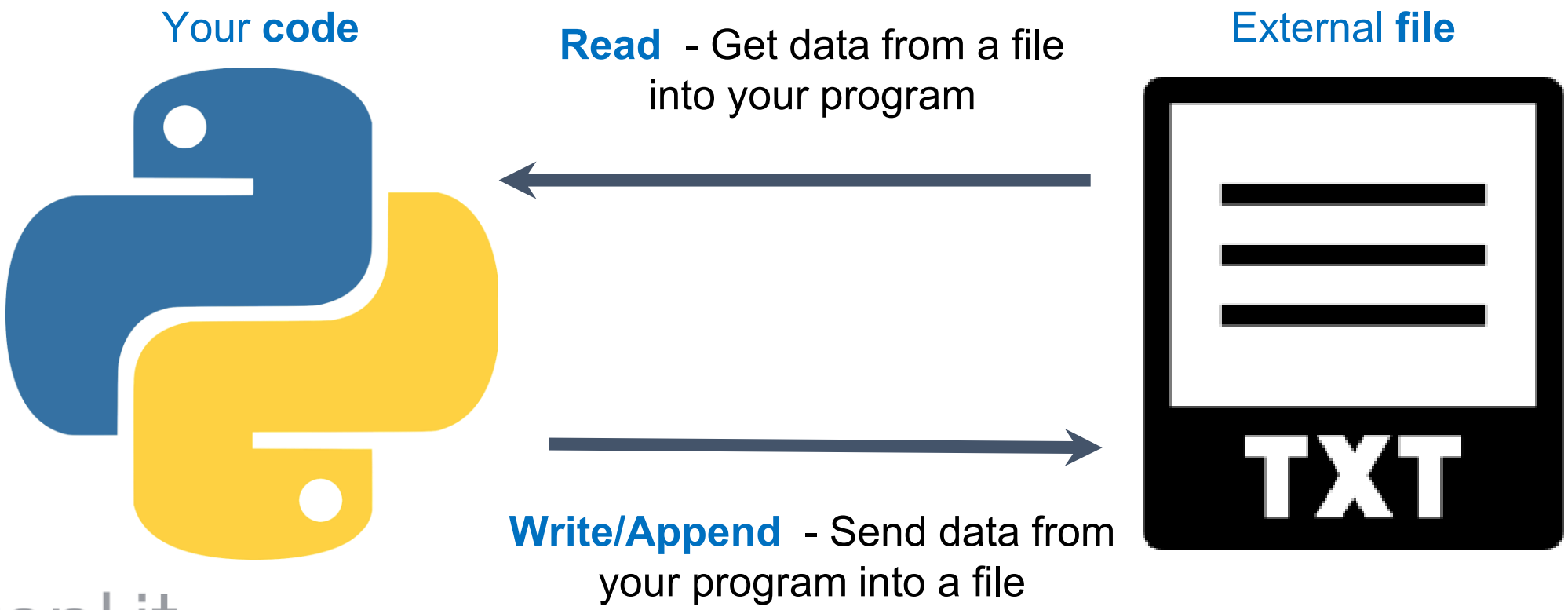
- **Opens** a file
- **Reads** data **from a file** into a program
- **Writes** data from a program **into a file**
- Appends data from a program into a file
- **Closes** a file

repl.it

Theory - File Handling

What Is File Handling?

Computer programs can import data from and export data to files outside the code.



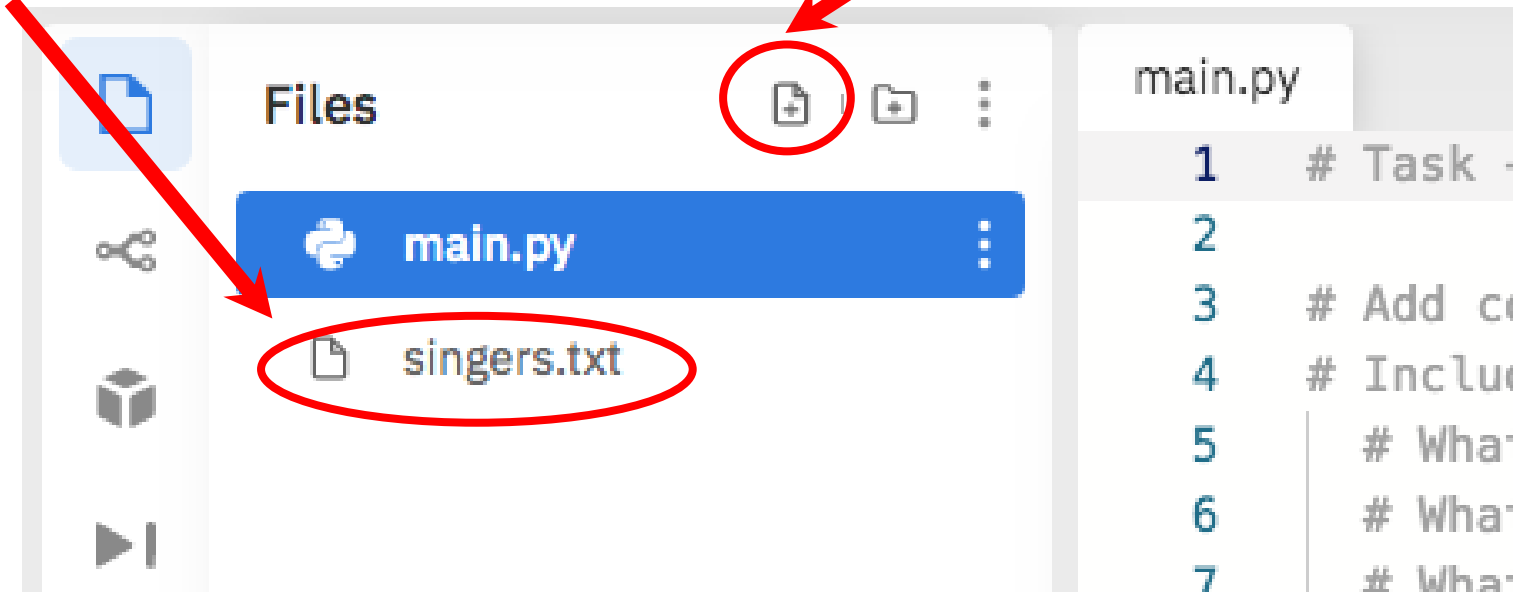
repl.it

Files In Replit

Replit allows you to upload and use external files with your code.

See the file in the main window.

Upload your own file (or just drag & drop)



repl.it

File Permissions

r	***Read*** - File can be 'looked at' by the program but not changed. Error if the file doesn't already exist.
a	***Append*** - File can be added to. File is created if it doesn't already exist.
w	***Write*** - Overwrites existing data, starting from the beginning of the file. File is created if it doesn't already exist.

repl.it

Read From a File

Read From A File - The Algorithm

1. Connect to and open the file
   a) Give the file name and path
   b) Set the permissions for opening

2. Read the contents into a variable

3. Output the variable

4. Close the file.

repl.it

Read All From A File - How To Code

1. Create a new variable to store the contents of the file.

2. 'open' tells the program to open the file.

4. 'r' means **read only.** The program can look at data from the file but not change it.

```
myFile = open("test.txt", "r")

for line in myFile:
 print(line)
myFile.close()
```

3. Put the **whole filename** in speech marks.

repl.it

Read From A File - How To Code

```
myFile = open("test.txt", "r")

for line in myFile:
 print(line)

myFile.close()
```

5. ‘for’ is another type of loop. It has a fixed length so does not need a condition.

6. ‘line in myFile’ sets the length of the loop to the number of lines in the external file.

7. Outputs each line from the file one by one. The loop moves through each one.

8. **ALWAYS** close the file once you have finished with it.

repl.it

Write & **Append** to a File

Write To A File - How To Code

1. Create a new variable to store the contents of the file.

2. 'open' tells the program to open the file.

4. 'w' means **write.** If there's an existing file with this name Python will open it. **If not it will create it.**

```
myFile = open("test.txt", "w")
myFile.write("Andy")
myFile.close()
```

3. Put the **whole filename** in speech marks.

5. Put the data or variable to be written

repl.it

Append To A File - How To Code

1. Create a new variable to store the contents of the file.

2. 'open' tells the program to open the file.

4. 'a' means **append.** This means 'add to the end' of the file.

```
myFile = open("test.txt", "a")

myFile.write("Andy")

myFile.close()
```

3. Put the **whole filename** in speech marks.

5. Put the data or variable to be written

repl.it

Tutorial II

Intermediate Python Programming

7 – Exception Handling.

Learning Goals/Objectives

Be able to read, comprehend, trace, adapt and create Python code that:

- Catches general **exceptions**
- Catches value exceptions
- Outputs suitable error messages

repl.it

What Is An Exception?

- An exception is something that happens during a program's **run that disrupts the flow of instructions.**
- They usually produce error messages - these are generated by an **exception handler.**
- It's possible to code our own exception handlers to prevent the program crashing in certain circumstances.

```
Enter a number100
Traceback (most recent call last):
  File "/Users/anh/www/150/online/examples/readingErrorMessages.py", line 14, in
 <module>
    y = x + 10
TypeError: Can't convert 'int' object to str implicitly
>>>
```

repl.it

Basic Exception Handling

Exceptions - How To Code

```
try:
    Run this code in normal circumstances
except:
  Run this code when there is an exception
```

repl.it

Exceptions - How To Code

```
name = “Dave”

try:
    print(name)
except:
  print(“The variable has not been assigned”)
```

repl.it

Exceptions - How To Code

```
name = "Dave"


try:
    print(name)
except NameError:
  print("The variable has not been assigned")
except:
    print("Something else went wrong)
```

repl.it

Exceptions - How To Code

```
name = “Dave”
try:
    print(name)
except NameError:
  print(“The variable has not been assigned”)
except:
    print(“Something else went wrong)
else:
    print(“nothing went wrong”)
```

repl.it

Exceptions - How To Code

```
name = “Dave”
try:
     print(name)
except NameError:
   print(“The variable has not been assigned”)
except:
     print(“Something else went wrong)
else:
     print(“Nothing went wrong”)
finally:
     print(“The try except has finished”)
```

repl.it

Value Errors

Value Errors - How To Code

1. Cast the input to the desired data type.

```
try:
    num1 = int(input("Type a number
    between 1 and 10"))
except ValueError:
        print("Hey, that wasn't a number!")
else:
        print("You typed " + num1)
```

2. Use '**ValueError**' in your except.

repl.it

Built-in Exceptions

Base Classes for Exceptions

See https://docs.python.org/3/library/exceptions.html for a full list so called „**base classes**“ of Python exceptions

- Some Examples

exception EOFError
Raised when the input() function hits an end-of-file condition (EOF) without reading any data. (N.B.: the io.IOBase.read() and io.IOBase.readline() methods return an empty string when they hit EOF.)

exception KeyboardInterrupt
Raised when the user hits the interrupt key (normally Control-C or Delete). During execution, a check for interrupts is made regularly. The exception inherits from BaseException so as to not be accidentally caught by code that catches Exception and thus prevent the interpreter from exiting.

exception TypeError
Raised when an operation or function is applied to an object of inappropriate type. The associated value is a string giving details about the type mismatch.

exception FileNotFoundError
Raised when a file or directory is requested but doesn’t exist. Corresponds to errno ENOENT.

exception NameError
Raised when a local or global name is not found. This applies only to unqualified names. The associated value is an error message that includes the name that could not be found.