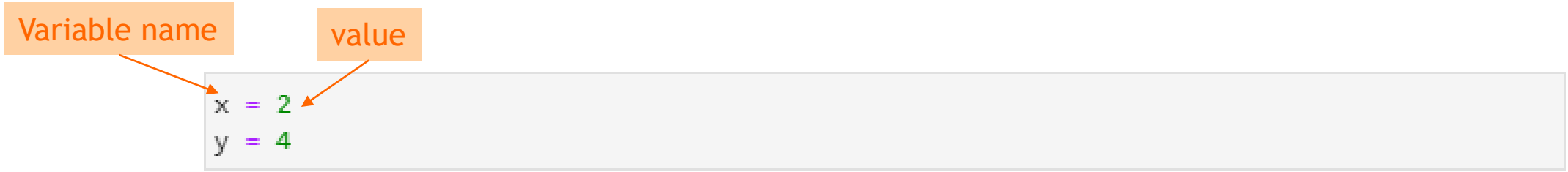
Variables and Data Types

Agenda

- Variables
- Data types
  - String
  - Number
- Operators
  - Math operator
  - Comparison operator
- Input and Output
- Error message

Variables

- **Variables:**
  - Variables are containers for storing data values.
  - Variables are used to access and manipulate data stored in memory.
  - A variable is created the moment you first assign a value to it.



Variables

- Variable Reassignment
  - Variables can reference different values while program is running
  - Variable that has been assigned to one type can be reassigned to another type

```
x = 2
print (x)
x = "hello world"
print (x)
```

```
2
hello world
```

Variable naming rules

- Rules for naming variables in Python:
  - Variable name cannot be a Python reserved word
    - class = “A001”
    - class_id = “A001”
  - Variable name cannot contain spaces
    - Item price = 150
    - Item_price = 150
  - First character must be a letter or an underscore; After first character may use letters, digits, or underscores
    - name1 = 30
    - 1name = 30
  - Variable names are case sensitive
    - WORD = “first”
    - word = “second”
- Variable name should reflect its use

Python reserved words

The following identifiers are used as reserved words, or keywords of the language, and cannot be used as ordinary identifiers. They must be spelled exactly as written here:

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

```
help("keywords")
```

Data types

Name	Type	Description	Example
String	str	A sequence of characters	“hello”, “course”, “covid-19”, “2”
Integer	Int	Whole numbers	2, 4, 100, 4000
Float	float	Numbers containing one or more decimals	3.8, 50.9, 100.0
Booleans	bool	Logical value indicating TRUE or FALSE	True, False
List	list	Ordered sequence of objects	[“hello”, “world”,”2021”] [“hello, 5, 100.0]
Dictionary	dict	Key: value pairs	{“key1”: name1, “key2”:name2}
Tuples	tup	Ordered immutable sequence of objects	(10,20) (“hello”, “world”)
Sets	set	Unordered collection of unique objects	{2,4,6,8} {3,”hello”, 50.9}

Types of values/variables

- Use `type` function to get the type of value

```
type("hello world")
```

```
str
```

```
type(100.1)
```

```
float
```

```
type("100.1")
```

```
str
```

- Check the type of variable

```
x = 10
type(x)
```

String

- Must be enclosed in single (') or double (") quote marks

```
str1 = 'hello world'
str2 = "hello world"
```

String – Indexing and slicing

- A string can be thought of as a list of characters.
- An index refers to a position within an ordered list.
- Each character is given an index from zero (at the beginning) to the length minus one (at the end).

h	e	l	l	o		w	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

```
# the length of a string
len(str1)
```

```
11
```

```
print(str1[0])
print(str1[0:2])
```

```
h
he
```

From position 0 to position 2, but NOT including 2.

Exercise

Exercise.A

(A.1) Define a variable with the name "mystr" and assign the string value "jupyter notebook" to this variable.

(A.2) Print the type of "mystr".

(A.3) Get the length of the variable "mystr" .

(A.4) Get the 5th character in "mystr".

(A.5) Get the first three characters in "mystr".

String - Methods

- Python has a set of built-in methods that you can use on strings

h	e	l	l	o		w	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

```
#converts the string to uppercase
str1.upper()
```

```
'HELLO WORLD'
```

```
# a specified values is replaced with a sepcified value
str1.replace("world","John")
```

```
'hello John'
```

```
# the number of times a specified value occurs in a string
str1.count("o")
```

```
2
```


You can use dir to list the methods available for an object.

String – Methods

- Use `find()` to search the place of a letter or a substring (it returns `-1` if it is not present).

```
# search the place of a letter/substring
str1.find("w")
```

```
6
```

```
str1.find("world")
```

```
6
```

```
str1.find("word")
```

```
-1
```

String – in

- Use the `in` operator to check if a string contains another string.

```
"world" in str1
```

```
True
```

```
"word" in str1
```

```
False
```

Exercise

Exercise.B

(B.1) Define a variable with the name "message" and assign the string value "Welcome to BI" to this variable.

(B.2) Convert 'message' to uppercase.

(B.3) Replace "BI" with "Oslo".

(B.4) Find the index position of the substring "to".

(B.5) Check if "message" contains the word "BI".

Numbers – integer and float

- Integer: Whole numbers
- Float: Numbers containing one or more decimals

```
x = 2
y = 5.7
print(type(x))
print(type(y))
```

```
<class 'int'>
<class 'float'>
```

Numbers – integer and float

- Type conversion

```
x = 2
y = 5.7
print(float(x)) # integer to float
print(int(y)) # float to integr (chop off the decimal portion of a number)
```

```
2.0
5
```

- Methods

```
# returns the absolute value of the given number
x = -9
abs(x)
```

```
9
```

```
# round a number
y = 21.9267
print(round(y))        (round to the nearest integer)
print(round(y,2))      (rounded to the second decimal place)
```

```
22
21.93
```


You can use dir(int) to list the methods available for int.

Operator – Math operators

- Operators are used to perform operations on variables and values.

Math operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
//	Floored division
%	Remainder

```
x = 5
y = 2
```

```
# operator "/" divides two number and returns a floating point value
x/y
```

```
2.5
```

```
# operator "/" divides two number and rounds the value down
x//y
```

```
2
```

```
# operator "%" returns the remainder left over when one operand is divided by a second operand
x%y
```

```
1
```

Operator – Comparison operators

- Comparison operators are used to compare values. It returns either True or False.

Comparison operator	Meaning
>	Greater than
<	Less than
==	Equal to
!=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to division

```
x = 5
y = 2
```

```
x > y
```

```
True
```

```
x == y
```

```
False
```

```
x <= y
```

```
False
```

String operation

- Some operators also work with strings.
  - Math operator (+, *)

```
word1 = "hello"
word2 = "world"
print (word1 + word2)
print (word1 * 3)
```

```
helloworld
hellohellohello
```

  - Comparison operator (==, !=)

```
word1 = "COVID-19"
word2 = "Covid-19"
print(word1 == word2)
print(word1.casefold() == word2.casefold()) # ignores cases when comparing
```

```
False
True
```

Exercise

Exercise.C

(C.1) Define a variable $n = 15.27391$. Print the type of n. **

(C.2) Round n to only two decimals.

(C.3) Divide n by 2.

(C.4) Use comparison operator to check if $n \leq 15$.

(C.5) Define a variable school = "BI". Use comparison operators to check whether the value of "school" is equal to "bi".

More operators

- Logical operator (We will see more details on the section *Conditional Expression*)
  - `and`, `or`, `not`

```
x = 5
y = 2
```

```
x == 5 and y ==2
```

```
True
```

```
x > 5 and y ==2
```

```
False
```

- Assignment operator (We will see more details on the section *Iterations and Loops*)
  - `=`, `+=`, `-=`, `*=`, `/=`

```
x = 2
```

```
x +=1
x
```

```
3
```

Input and Output

Input

- Most programs need to read input from the user. The simplest way to accomplish this in Python is with input().

```
name = input('Enter your name: ')
print( "Hi, ", name)
```

```
Enter your name: John
Hi,  John
```

- input() always returns a string. If you want a numeric type, then you need to convert the string to the appropriate type.

```
x1 = int(input('Enter a number: '))
x2 = int(input('Enter a number: '))
print ('The sum of the two numbers you have entered is:',x1+x2)
```

```
Enter a number: 6
Enter a number: 3
The sum of the two numbers you have entered is: 9
```

Exercise

Exercise.D

(D.1) Define an input box named course_id and print out the following message.

Expected result:

```
Enter course ID: EBA3400
You have registered for the course EBA3400
```

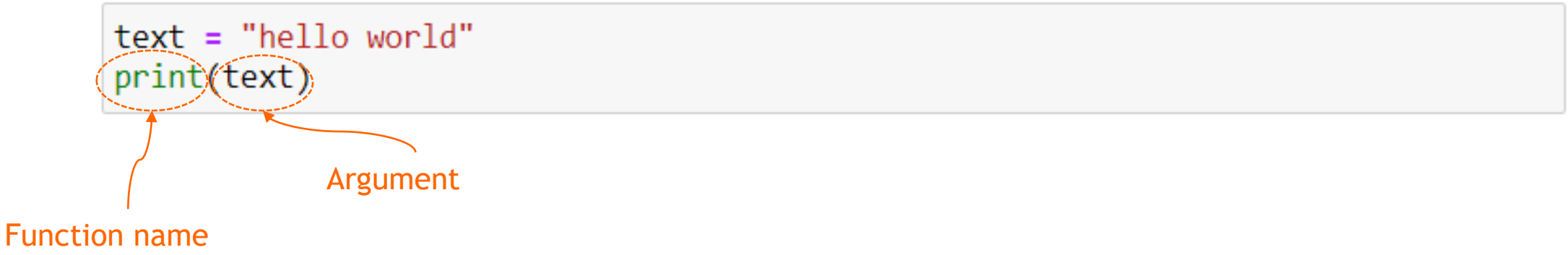
(D.2) Define an input box named number and check if number > 0.

Expected result:

```
Enter a number: 10
True
```

Output – print function

- Display output with the `print` function
  - Function: piece of prewritten code that performs an operation.
  - Argument: data given to a function



Output – print multiple items

- Displaying multiple items with the print function
  - Items are separated by commas when passed as arguments
  - Items are automatically separated by a space when displayed on screen

```
# items are automatically separated by a space
ctry1 = "Iceland"
ctry2 = "Portugal"
ctry3 = "Finland"
print("Countries with a low risk", ctry1, ctry2, ctry3)
```

```
Countries with a low risk Iceland Portugal Finland
```

  - Argument “sep” and “end”

```
# argument “sep” and “end”
ctry1 = "Iceland"
ctry2 = "Portugal"
ctry3 = "Finland"
print("Countries with a low risk", ctry1, ctry2, ctry3, sep = ", ", end = ".")
```

```
Countries with a low risk, Iceland, Portugal, Finland.
```

Output – print multiple lines

- Displaying multiple lines with the print function
  - Break a statement into multiple lines by multiline continuation character (\n)

```
print ("There were 315 confirmed cases in Norway.\nThe R rate is 1.0.")
```

```
There were 315 confirmed cases in Norway.
The R rate is 1.0.
```

- Try to use “sep” to print multiple items by line.

```
ctry1 = "Iceland"
ctry2 = "Portugal"
ctry3 = "Finland"
print("Countries with a low risk", ctry1, ctry2, ctry3, sep = "\n")
```

```
Countries with a low risk
Iceland
Portugal
Finland
```

Exercise

(E.1) Define three variables, n1 = 30, n2 = 08, n3 = 2021. Print all variables in one line.

Expected result:

```
30 08 2021
```

```

```

(E.2) Use the variables defined in (E.2). Print all variables in one line and connect them with "-".

Expected result:

```
30-08-2021
```

```

```

(E.3) Try to modify the following code to print sentences line by line.

Expected result:

```
Solve complex business problems with analytical thinking.
Work alongside data scientists to find new insights.
Become part of a promising new field aiming to disrupt traditional banks and financial institutions.
```

```
print("Solve complex business problems with analytical thinking. Work alongside data scientists to finc
```

```
Solve complex business problems with analytical thinking. Work alongside data scientists to find new
insights.Become part of a promising new field aiming to disrupt traditional banks and financial insti
tutions.
```

Output - String formatting

- String formatting is the process of inserting a custom string or variable in predefined text.

- Python uses two different styles of string formatting.
  1) str.format() method
  2) % string formatter (optional)

Output – str.format() method

- The brackets and characters within them (called format fields) are replaced with the objects passed into the str.format() method.

```
case_num = 315
country = "Norway"
print("There were {} confirmed cases in {}.".format(case_num, country))
```

```
There were 315 confirmed cases in Norway.
```

```
r = 0.9684
print ("The interest rate is {}.".format(r))
```

```
The interest rate is 0.9684
```

- Use `:.2f` to format a float to two decimal places.

```
r = 0.9684
print ("The interest rate is {:.2f}.".format(r))
```

```
The interest rate is 0.97.
```

Output - Old % str formatter

- The % operator can also be used for string formatting.

```
case_num = 315
country = "Norway"
print("There were %d confirmed cases in %s." %(case_num, country))
```

```
There were 315 confirmed cases in Norway.
```

```
r = 0.9684
print ("The interest rate is %f" %r)
```

```
The interest rate is 0.968400
```

```
r = 0.9684
print ("The interest rate is %.2f" %r)
```

```
The interest rate is 0.97
```

Exercise

(F.1) Define two variables, x = 5.5, y = 20. Use format() to print the following text.

Expected result:

```
The first number is 5.5 and the second number is 20.
```

(F.2) Use %formatter to print the same text.

Types of errors

1) **Syntax errors** : Violation of grammar" rules. Easiest to fix. Python tells you at which line of your code the first error is detected

```
print "hello world"
```

```
  File "<ipython-input-2-6d29d8fb337c>", line 1
    print "hello world"
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("hello world")?
```

2) **Logic errors** (also called semantic errors): logical errors cause the program to behave incorrectly. A program with logic errors can be run, but it does not operate as intended.

```
x = float(input('Enter a number: '))
y = float(input('Enter a number: '))

z = x+y/2
print ('The average of the two numbers you have entered is:',z)
```

```
Enter a number: 3
Enter a number: 4
The average of the two numbers you have entered is: 5.0
```

Built-in exceptions

- Even if the syntax of a statement is correct, it may still cause an error when executed.
- There are some built-in exceptions in Python.

```
#Type error
str1 = "50"
str1/2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-94baa66c9975> in <module>
      1 #Type error
      2 str1 = "50"
----> 3 str1/2

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

```
#Name error
print(new_str)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-13-55139cfbc8ef> in <module>
      1 #Name error
----> 2 print(new_str)

NameError: name 'new_str' is not defined
```

```
# Zero division error
10/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-14-1c59a852d30a> in <module>
      1 # Zero division error
----> 2 10/0

ZeroDivisionError: division by zero
```

See more types of errors https://docs.python.org/3/library/exceptions.html

Debugging

- What can you do about the errors? Debugging
  - Debugging is the process of finding and resolving bugs (problems that prevent correct operation) within computer programs.

- **Think**:
  - What kind of error is it: syntax error, logic error?
  - What information can you get from the error messages, or from the output of the program?
  - What kind of error could cause the problem you're seeing?
  - What did you change last, before the problem appeared?

- **Retreat**:
  - At some point, the best thing to do is back off, undoing recent changes, until you get back to a program that works and that you understand. Then you can start re-building.