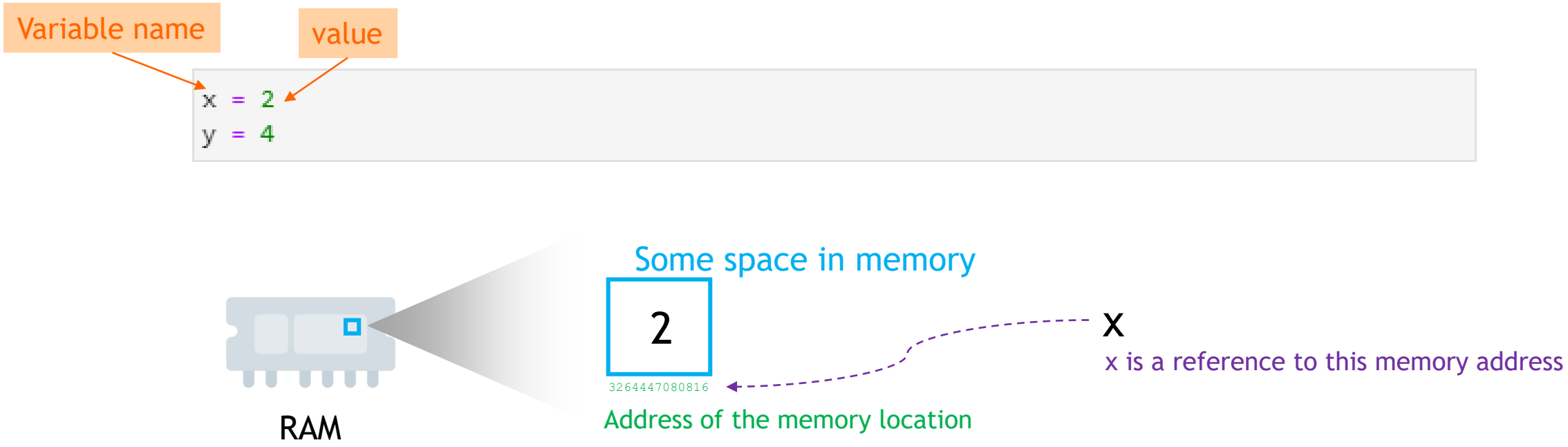
Variables and Data Types

Agenda

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

Variables

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



- RAM is short for “random access memory”.
- RAM is the main memory of a computer system.

Variables

- Variable reassignment
  - Variables can reference different values while program is running.

```
x = 2
print (x)

x = 3
print (x)
```

```
2
3
```

  - Variable that has been assigned to one type can be reassigned to another type.

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

```
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.
    - 1name = 30 ✗
    - name1 = 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

- 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	→ character
0	1	2	3	4	5	6	7	8	9	10	→ Index

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

```
11
```

```
print(str1[0])
print(str1[4])
```

```
h
o
```

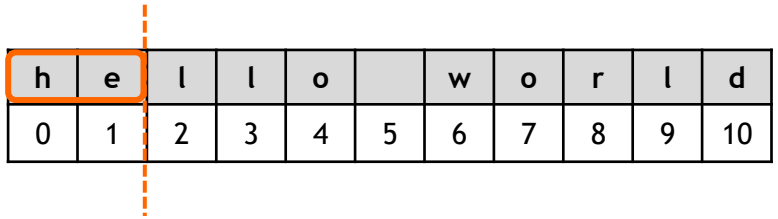
String – slicing

- Specify the start index and the end index, separated by a colon, to get a part of the string.
- Format: `[ start_index : stop_index ]`

```
str1[0:2]
```

```
'he'
```

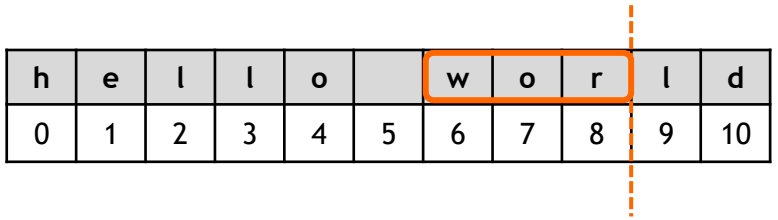
From position 0 to position 2, but NOT including 2.




```
str1[6:9]
```

```
'wor'
```

From position 6 to position 9, but NOT including 9.



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 substring `note` from the variable `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
```


- Python method is like a function, but it is associated to an object (data type).
- Strings are immutable, so you cannot change the contents of string variables.
- A possible solution is to create a new string variable with the necessary modifications.

String – Methods

- Use `find()` to find the first occurrence of the specified value.

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

```
# search for the position of the letter
str1.find("w")
```

```
6
```

```
# search for the position of the substring
str1.find("world")
```

```
6
```

- It returns `-1` if the value is not found.

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

```
-1
```

BI

- Use dir(str) to list the methods available for string.

String – in

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

```
str1 = "hello world"
```

```
"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 all letters in `message` to uppercase.

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

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

(B.5) Check if the following variable contains the word `data`.

```
book = "Python for Data Analysis"
```

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

- Functions for numbers

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

```
9
```

```
# round a number
y = 21.9267
print(round(y))
print(round(y,2))
```

(round to the nearest integer) 21.9267

(rounded to the second decimal place) 21.9267

```
22
21.93
```

- Type conversion

```
# float to integer
print(int(y))
```

(chop off the decimal portion of a number) 21.9267

```
21
```

```
# integer to flaot
print(float(x))
```

```
-9.0
```

Convert string to integer

- Convert string to integer and assign the result to a new variable.

```
z1 = "20"
print(type(z1))
```

```
<class 'str'>
```

```
z2 = int(z1)
print(type(z2))
```

```
<class '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()) #ignore case when comparing
```

```
False
True
```

Exercise

Exercise.C

(C.1) Define a variable `n = 15.27391`. Print the data type of `n`.

(C.2) Round `n` to two decimal places.

(C.3) Divide `n` by `2`.

(C.4) Use comparison operator to check if `n ≤ 15`.

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()` to create an input box.

Define a variable named “y" whose value is given by the user.

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

```
Enter your name:James

Hi James
```

Input

- 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) Write a program that asks user to enter the course ID and prints out the following message based on the entered value.
Expected result:

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

```
```

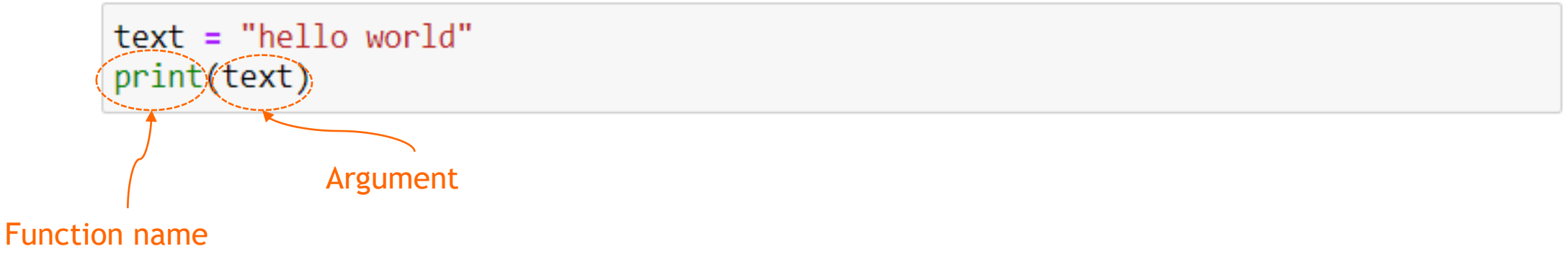
(D.2) Write a program that asks the user to enter their first name and last name, and prints out their initals.

Expected result:
First name: `James`
Last name: `Smith`
`JS`

```
```

Output – print function

- Display output with `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.
  - Split text 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

Exercise.E

(E.1) Given the following variables. Print the values of these variables on one line.

Expected result:

```
30 08 2021
```

```
dd = "30"
mm = "08"
year = "2022"
```

```

```

(E.2) Use the variables defined in (E.1). Print the values of these variables on one line and connect them with the symbol `-`.

Expected result:

```
30-08-2021
```

```

```

Output - String formatting

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

```
There were _____ confirmed cases in _______.

There were _225_ confirmed cases in __Oslo__.
There were _160_ confirmed cases in _Bergen_.
```

- Different ways of string formatting in Python:
  1) `f-strings`
  2) `str.format()`
  3) `%` string formatter (optional)

Output – (1) f-strings

- Format a string by simply prefixing it with the letter "f“, and specify variables (or expression) in curly brackets.

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

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

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

```
The interest rate is 0.9684.
```

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

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

Format a float to two decimal places

- 'fstring' is short for 'formatted string’.
- ‘fstrings’ was introduced in Python 3.6

Output – (2) str.format()

- 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
```

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

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

Output – (3) % 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

Exercise.F

(F.1) Write a program that asks the user to enter their name and prints out the following text. (Use fstrings)
Expected result:
Enter your name: `Max`
Hi Max, welcome to BI!

(F.2) Write a program that asks the user to enter two numbers. Prints out the sum of these variable in the following format.
The sum of __ and __ is ___.

First number: `10`
Second number: `5`
The sum of 10 and 5 is 15.

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.