PYTHON COURSE

ENTRY LEVEL

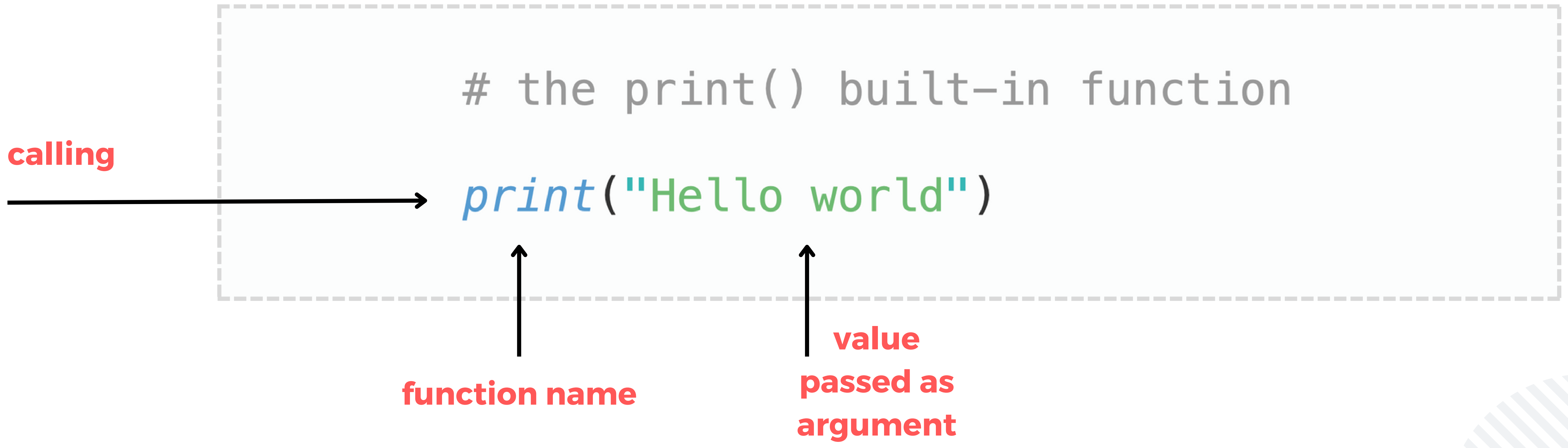
Basics of programming in Python 3.10

This course will cover part of the arguments found in PCEP™ – Certified Entry-Level Python Programmer Certification

INTRO TO FUNCTIONS AND METHODS

WHAT FUNCTIONS ARE

- Functions are **block of codes that only runs when called in the program**
- You can perform actions using functions, **passing values as parameters**
- Functions can have a “result” or not, in case, **the “result” of a function is called return value**
- You can assign functions’ return values to variable



```python
# the print() built-in function

print("Hello world")
```

ARGUMENTS

You can **pass arguments into functions in two ways** (or mixed)

- positional arguments
- key-word syntax

POSITIONAL

```python
# simple positional arguments passing

print("Hello world")
```

ARGUMENTS

- **KEY-VALUE**

passing **key-value arguments** requires **sintax:** parameter exact name = value to assign

- **MIXED**

mixed arguments parring require key-value arguments to follow all positional arguments

```python
# simple positional arguments passing

print("Hello world", end="")
```

parameter name

value passed as argument

PARAMETERS' DEFAULT VALUES

some functions' parameters can have default values stored, so, if not passed as arguments when the function is called (overriding their values), they will be used for the function to execute.

in order to change them, you need to pass a value for those parameters

INPUT/OUTPUT OPERATIONS

- **print() function**

you can pass multiple positional arguments to the print function, all will be concatenated in a single String, and printed

```python
# multiple positional arguments print

animal = "Monkey"
ambient = "Jungle"

print(animal, ambient)
# Monkey Jungle
```

- **end and sep key-value parameters**

```python
# multiple positional arguments print
# overriding sep and end key-value parameters

animal = "Monkey"
ambient = "Jungle"

print(animal, ambient, sep="..", end="--")
# Monkey..Jungle--
```

- Multiple values passed as positional arguments will be concatenated with sep parameter value in between each other (default value = “ “)

- Aftwer all the value, the end parameter’s value will be concatenated (default value = “\n”)

Example of overriding end parameter’s default value

```
name = "Scott"

print(name) # end default value \n will be concatenated at the end of name
print(name)

# Scott
# Scott

print(name, end = " ") # " " will be concatenated at the end of name
print(name)

# Scott Scott
```

INPUT/OUTPUT OPERATIONS

- **input() function**

the input function **consents obtaining data from user input**

- input function **has a return value** (so this value can be stored in a variable)
- Accepts one positional argument, it will be printed right before asking user input

```
name = input("Insert your name: ") # return type is a string

print("Hello,", name)
```

```
[> python3 input.py
 Insert your name: Madelen
 Hello, Madelen
```

CASTING

Casting is the operation of **changing type to a value**. It **can be automatic (implicit) or manual (explicit)**

Casting is **used to be able performing certain operations** without incurring in a TypeError (operations or actions between unsupported values)

- es: if you want to concatenate a numeric to a string
- if you want to multiply a number in string form with a numeric

CASTING: IMPLICIT

Python automatically cast value types in order to perform some operations

```
a = 10
b = 5

c = 10 / 5
d = 10 + 0.0

print(type(c)) # float
print(type(d)) # float
```

EXPLICIT CASTING WITH FUNCTIONS

Python provides some built-in functions for type casting of built-in data types

- **int(), str() and float() functions**

passing a variable or a value to those functions, will result in a **type change of the value stored in that variable**

all of this type of function **have return type** (return type depends on the function)

```
a = str(1995)
b = 1995
c = float(1995)
d = int("1995")

print(type(a), type(b), type(c), type(d)) # str int float int
```

CASTING: EXAMPLES

Casting a **String** to **Integer** in order to sum it with a Numeric

```
a = "1995"

b = int(a) + 5
```

Casting an **Integer** to **String** to concatenate it

```
a = 1995

b = "Born in " + str(a)
```

CASTING: EXAMPLES

pay attention when casting float to int, decimal part will be removed without approximation

```
a = 1995.9
b = int(a)

print(b) # 1995
```

BUILT-IN FUNCTIONS: LEN()

- len() function is used to **check the length of some objects**
- it has a **return value of type int**, representing the **number of elements in the object**
- it must be used on iterable objects

- **len() function with strings**

```
a = "hello"
b = len(a)

print(b) # will print 5, the number of single characters in the string
```

BUILT-IN FUNCTIONS: ISINSTANCE()

- used to check if an object is referrable to a certain class
- has boolean return type

```
name = "Norman"

# returns true if first argument is an object of type as second argument
result = isinstance(name, str) # True cause name is a string
```