CS 38003 PYTHON PROGRAMMING

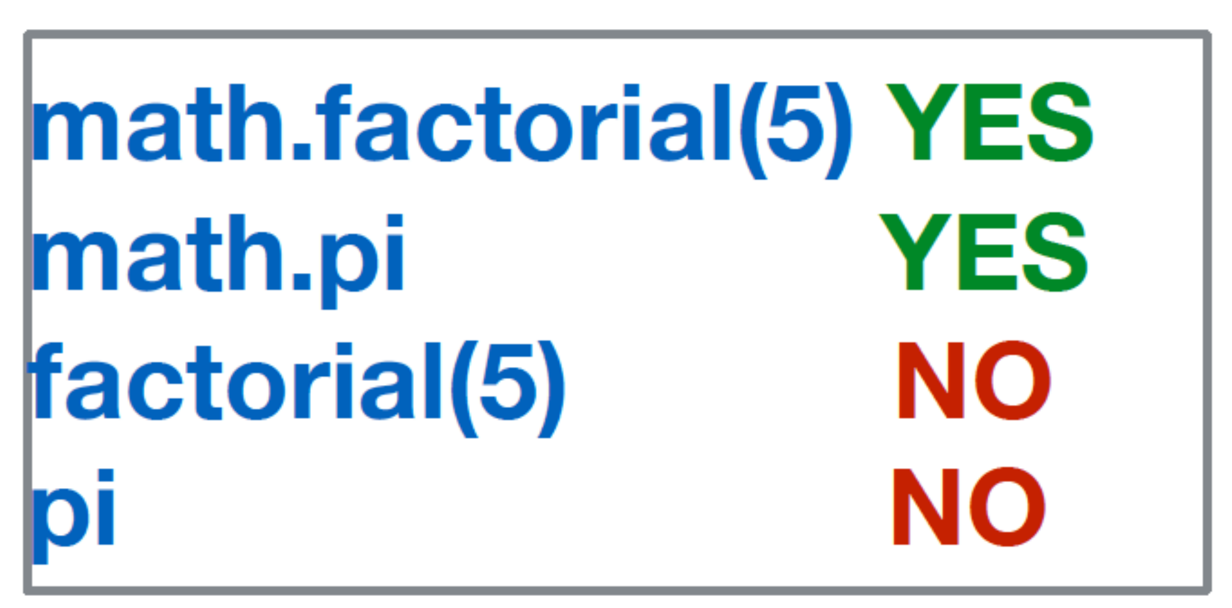
Ruby Tahboub

IMPORTING LIBRARIES

IMPORTING A LIBRARY

- There are three ways to import a library/ module

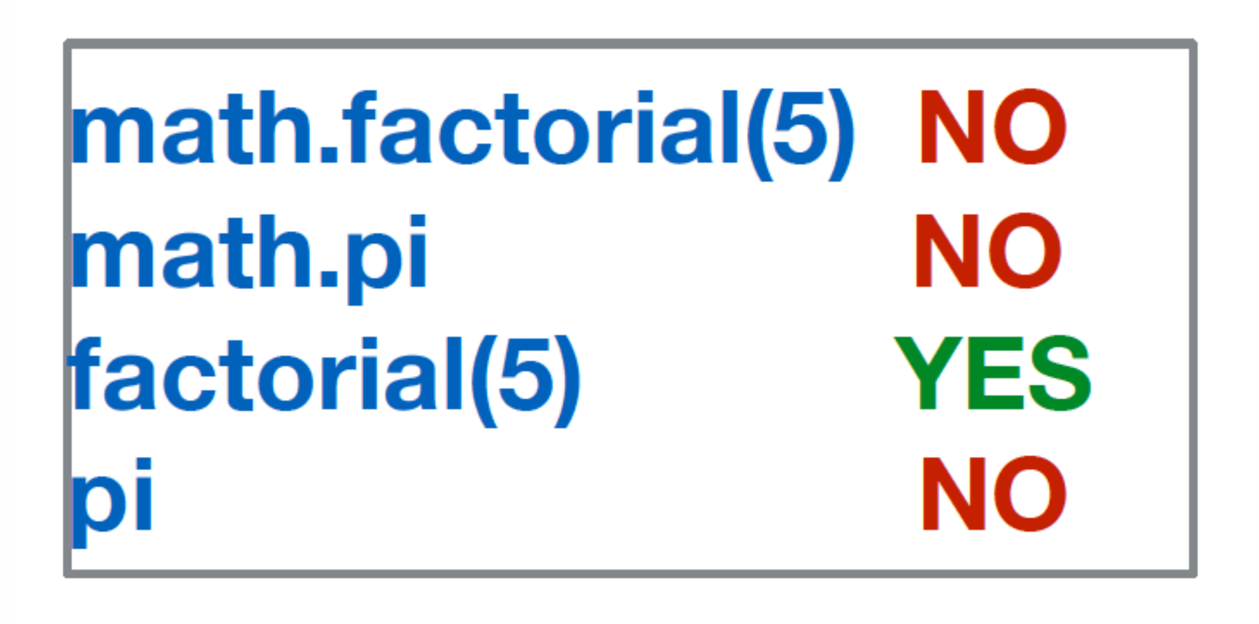
1. import math
2. from math import factorial
3. from math import *



math.factorial(5)	YES
math.pi	YES
factorial(5)	NO
pi	NO

IMPORTING A LIBRARY

- There are three ways to import a library/ module
  1. import math
  2. from math import factorial
  3. from math import *

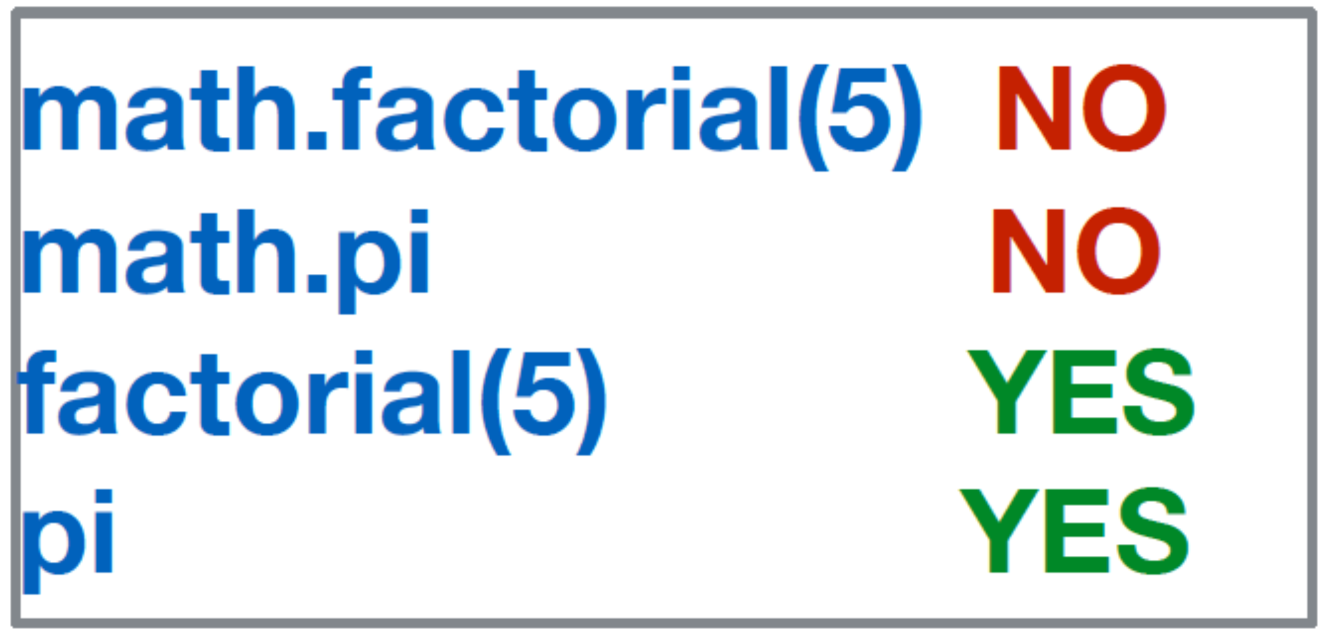


math.factorial(5)	NO
math.pi	NO
factorial(5)	YES
pi	NO

IMPORTING A LIBRARY

- There are three ways to import a library/ module

1. import math
2. from math import factorial
3. from math import *



math.factorial(5)	NO
math.pi	NO
factorial(5)	YES
pi	YES

MATH LIBRARY

- `math.pow(x, y)` Returns x raised to the power y.
- `math.sqrt(x)` Returns the square root of x.
- `math.factorial(x)` Returns x factorial.
- `math.ceil(x)` Returns the ceiling of x.
- `math.floor(x)` Returns the floor of x.
- `math.log(x, y)` Returns the log of x to the base y. If y is not provided then the natural logarithm (base e) is used.
- `math.sin(x)`, `math.cos(x)` … all trigonometric functions. The angle x should be in radians.
- `math.degrees(x)` - convert x from radians to degrees
- `math.radians(x)` - convert x from degrees to radians
- `math.pi` - the mathematical constant pi π
- `math.e` - the mathematical constant e

FUNCTIONS

DEFINING FUNCTIONS

- Function definition begins with **`def`** and ends with a colon.
- Indentation matters.
- Input parameters are optional.
- No function header.

```
def function_name(parameters):
  Line 1
  Line 2
  ...
  return value1, value2, ...
```

PARAMETERS

- Parameters in Python are `call by assignment`.
  - This sometimes acts like call by reference (for mutable datatypes) and sometimes like call by value (for immutable datatypes).
  - All assignment in Python, including binding function parameters, uses reference semantics.

parameters

```
def multiply(x, y):
  return x * y

>>>print (multiply(10,5))
50
```

ARGUMENTS

- An argument is the actual value of the function parameter when it is passed.
- A function argument can be:
  - Value `multiply(10,5)`
  - Expression `multiply(10+x,5/y)`
  - Variable `multiply(a,b)`
  - Function call `multiply(multiply(10,5),2)`
  - …

```
def multiply(x, y):
  return x * y

>>>print (multiply(10,5))
50
```

arguments

DEFAULT ARGUMENTS

- The arguments of a function may have default values.
- Functions can be called using parameter names e.g., `multiply (x=10, y=5)`

```
def multiply(x=10, y=20):
  return x * y

>>>print (multiply(10,5))
50

>>>print (multiply(3))
60

>>>print (multiply())
200

>>>print (multiply(x=5, y=3))
15
```

RETURN VALUE

- All functions in Python have a return value, even if no return line inside the code.
- Functions without a return value, return the special value None.
  - None is a special constant in the language.
  - None is used like `NULL`, `void`, or `nil` in other languages.
  - None is also logically equivalent to False.
  - The interpreter doesn't print None.

```
def multiply(x, y):
  return x * y

>>>print (multiply(10,5))
50
```

THE MAIN FUNCTION

- Python does not have a main function where execution starts as C or Java.
- Execution in Python Starts at level 0 indentation.
- In this class, we will create our own `main` function whenever needed.

```
def factorial(n):
  prod = 1
  for i in range(1, n + 1):
    prod = prod * i
  return prod

def main():
  print (factorial(5))

main()
```

```
120
```

EXCEPTION HANDLING

EXCEPTION HANDLING

- Handles errors that cause programs to produce incorrect results or terminate the program unexpectedly i.e., runtime errors.
- Examples:
  - Division by zero.
  - Opening a file that does not exist.
  - …

```
try:
  <body>
except <ErrorType1>:
  <handlerErrortype1>

except <ErrorType2>:
  <handlerErrortype2>

except: <handlerDefault>

else:
  <elseBlock>

finally:
  <finallyBlock>
```

EXCEPTION HANDLING SYNTAX

- Python executes the code in the `try` block.
- If an exception is raised, Python looks for a matching `except` condition.
- There can be multiple `except` blocks.
- There can be a default `except` block

```
try:
  <body>
except <ErrorType1>:
  <handlerErrortype1>

except <ErrorType2>:
  <handlerErrortype2>

except: <handlerDefault>

else:
  <elseBlock>

finally:
  <finallyBlock>
```

EXCEPTION HANDLING SYNTAX

- If no exception is raised, Python executes the `else` block.
- At the end, Python executes the `finally` blocks which typically used for clean up or undo previous changes.
- Python always executes the `finally` block.
- `else` and `finally` blocks are optional.
- There should be at least one `except` block.

```
try:
  <body>
except <ErrorType1>:
  <handlerErrortype1>

except <ErrorType2>:
  <handlerErrortype2>

except: <handlerDefault>

else:
  <elseBlock>

finally:
  <finallyBlock>
```

EXAMPLE

```
try:
  a = 1/0
except ZeroDivisionError:
  print ("ZeroDiv")
except:
  print ("DefaultExcep")
else:
  print ("Else")
finally:
  print ("Finally")
```

```
ZeroDiv

Finally
```

THE ROOTS OF QUADRATIC EQUATION

```
import math

try:
  a = eval(input("Please enter the co-efficient a "))
  b = eval(input("Please enter the co-efficient b "))
  c = eval(input("Please enter the co-efficient c "))
  discrim = math.sqrt(b*b - 4*a*c)
  root1 = (-b + discrim) / (2*a)
  root2 = (-b - discrim) / (2*a)
  print ("The first root is", root1)
  print ("The second root is", root2)
except ZeroDivisionError:
  print ('division by zero')
except ValueError:# square root of a negative number
  print ('-ve under the root')
except:
  print ('Cannot find roots')
finally:
  print ("Finally")
```

Quadratic Formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
1

10

1

The first root is -
0.10102051443364424The
second root is -
9.89897948556356

Finally
```

THANK YOU!