ERROR AND EXCEPTION HANDLING

PREPARED BY: LUIS MEING

OBJECTIVES:

01

Python Code Blocks and Keywords

1.1: Code Block Definition
1.2: Importance
1.3: Keywords

02

Python Exception Handling

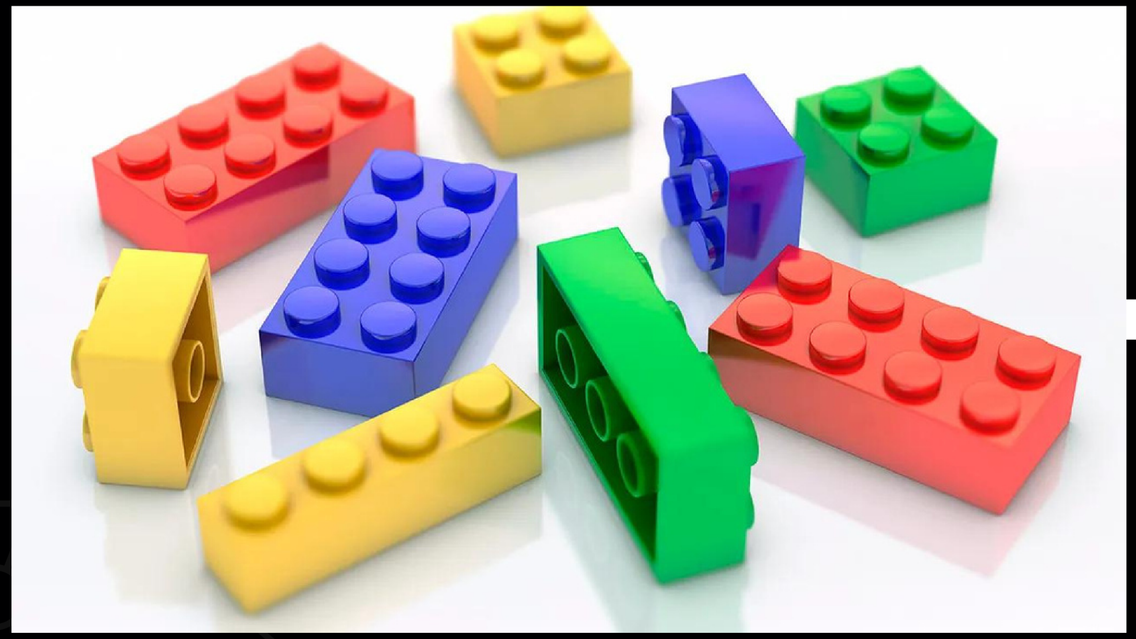
2.1: Definition
2.2: Try ... Except Block
2.3: Try ... Else Block
2.4: Try ... Finally Block
2.5: Conclusion

Code Block

- A block is a piece of Python program text that is run as a unit.
- A block is a smaller component of your program

1.1 Code Block Definition

Commands, variables, etc



Code Blocks

1.1 Code Block Definition

Code Blocks

Programs

Code Block

- Imagine it as beginning of a bigger command and the end
- Code blocks are identified by their indentation in Python.

1.1 Code Block Definition

```
print ("Hello")
```

1.1 Code Block Definition

```
num1 = input("first: ")
num2 = input("second: ")


print(int(num1)+int(num2))
```

1.1 Code Block Definition

```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

1.1 Code Block Definition

```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

is a code block inside a bigger code block

is a code clock

1.1 Code Block Definition

```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

Will run first

1.1 Code Block Definition

```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

Then this

1.1 Code Block Definition

```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

Will run first

1.1 Code Block Definition

```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

Then this

1.1 Code Block Definition

```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

Then this

What can we conclude?

- In Python, the code is sequential
- It is modelled after a waterfall

What can we conclude?

- The code is read from top to bottom, from major code block to a minor code block one at a time

1.2 Importance

```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

What if we don't indent?

1.2 Importance

```
print ("Good day!")
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

1.2 Importance

```
Good day!
first: 23
second: 27
50
```

```
Good day!
first: 6
second: one
Invalid Input! Try again
```

1.2 Importance

```
print ("Good day!")
try:
num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

1.2 Importance

```
C:\Users\Dell\Desktop\MyPythonPrograms>test2.py

  File "C:\Users\Dell\Desktop\MyPythonPrograms\
test2.py", line 3
    num1 = input("first: ")
    ^
IndentationError: expected an indented block af
ter 'try' statement on line 2
```

1.2 Importance

```
for a in range(1,n):
    for b in range(a,n):
        c_square = a**2 + b**2
        c = int(sqrt(c_square))
        if ((c_square - c**2) == 0):
            print(a, b, c)
```

*Blocks are a part of Python's syntax

1.3 Keywords

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

1.3 Keywords

- Never use keywords or commands to name your variables.

Exception

- It is an unwanted or unexpected event when a computer program runs.

Exception Handling

- It is the process of responding to unwanted or unexpected events when a computer program runs.

Exception Handling

- When exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

Try...

- In Python, exceptions can be handled using a "try" statement.
- The critical operation which can raise an exception is placed inside the try clause.

Try...

```
try:
        num1 = input("first: ")
        num2 = input("second: ")


        print(int(num1)+int(num2))
```

... Except

- The code that handles the exceptions is written in the except clause.
- We can thus choose what operations to perform once we have caught the exception.

... Except

```
except:
    print("Invalid Input! Try again")
```

Try... Except

```
1  try:
2      num1 = input("first: ")
3      num2 = input("second: ")
4
5      print(int(num1)+int(num2))
6  except:
7      print("Invalid Input! Try again")
```

Try... Except

- Creates a block of code
- "Try " indents those proceeding after it until "Except" or another one of its optional keyword pair is used

Try... Else

- you can use the optional "else" keyword with the try... except statement
- you may use this if you want code to run after the "try" statement did not have an error

Try... Else

- you can use the optional "else" keyword with the try... except statement
- you may use this if you want code to run after the "try" statement did not have an error

Try... Else

```
try:
    num1 = input("first: ")
    num2 = input("second: ")
except:
    print("Invalid Input! Try again")
else:
    print(int(num1)+int(num2))
```

Try ... Finally

- The try statement in Python can have an optional finally clause.
- This block is executed no matter what, and is generally used to release external resources.

Try ... Finally

```
try:
    num1 = input("first: ")
    num2 = input("second: ")
    f = open("test.txt")
except:
    print("Invalid Input! Try again")
else:
    print(int(num1)+int(num2))
finally:
    f.close()
```

Syntax:

```
try:
        <content>
except:
        <what happens when exceptions occur>
<else:>
        <code to run if no exception in try>
<finally:>
        <code to run even if exception is done in try>
```

? Activity!~

Please bring out 1/8 yellow paper, write your name and section.

Write only what is asked. Numbering is allowed.

? Activity!~

Write the output of the following codes. If it is an error, state whether it is syntactic or semantic. If semantic, write the result as well. If none, state as such.

1 Activity!~

```
print = 23
print2 = 32


print("23+32=" + (print+print2))
```

2 Activity!~

```
print('3+3**3/(9*2)*4')
```

3 Activity!~

If I try to input 23 and 32 respectively

```
try:
        num1 = input("first: ")
        num2 = input("second: ")
else:
        print(int(num1)+int(num2)
```

4 Activity!~

If I try to input 23 and 32 respectively and want to know the sum

```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1+num2))
except:
    print("Invalid Input! Try again")
```

5 Activity!~

If I try to input 2 and 4 respectively

```
try:
    num1 = int(input("first: "))
    num2 = int(input("second: "))

    print("Answer is: " + (num1/num2**(4/2)+2*2))
    print("Answer is: " + (num1/num2**(4/2*2)+2*2))
    print("Last is: " + (num2/num1**num2))

except
    print("Invalid Input! Try again")
```

? Activity!~

1. error - syntactic
2. `3+3**3/(9*2)*4`
3. error - syntactic
4. error - semantic - 2332
5. error - syntactic