LECTURE 03

Friday, August 16, 2024 1:40 PM

EXCEPTIONS

Syntax Errors

These are entirely up to you to fix from the start, e.g.

```
print("hello word"
```

error: unterminated string literal

There can also be run-time errors.
- can occur from faulty or bad input.

```
x = int(input("What is x: ?")
print(f"x is {x}")
```

If a user inputs a number, nothing is wrong. But what happens if a user doesn't input an it?
e.g, Let's type in "cat":

error: "ValueError: invalid literal for int() with base 10: 'cat'"

the literal is the user input.

You wanna write code with error handling.

"try" keyword

```
try:
.... x = int(input("What is x: ?")
.... print(f"x is {x}")
except ValueError:
.... print("x is not an integer")
```

Plenty of error types, but this one helps.
It's bad practice to try and encompass every type of erro.
- unfortunately, it's not always obvious which error you should choose.

Try to only try-catch the code that might raise the error. But do it correctly.

```
try:
.... x = int("input: what is x? ")
except ValueError
.... print("x is not an integer.")

print(f"x is {x}")
```

- when we input an int, no prob.
- but we get a NameError if it's not an int.

"NameError: `x' is not defined"
consider what happens when a string gets passed to the int() function.
- we catch the ValueError, so x does not actually get defined.

We will leverage 'else'

```
1 try:
2 .... x = int("input: what is x? ")
3 except ValueError
4 .... print("x is not an integer.")
5 else:
6 .... print(f"x is {x}")
```

This fixes the error. If string is inputted, we catch the error. Now, we ensure to only execute line 6 as it has a value

Be more "friendly"

Give the user a chance to fix mistake.
How? Loops!

```
while True:
.... try:
........ x = int(input("What's x? "))
.... except ValueError:
........ print("x is not an integer")
.... else:
........ break
```

← we escape this while-loop with "break"!

```
print(f"x is {x}")
```

Slightly more efficient...

```
while True:
.... try:
........ x = int(input("What's x? "))
........ break
.... except ValueError:
........ print("x is not an int")

print(f"x is {x}")
```

Let's write a function, get_int() that does this work.
- remember we return now!

```
def main():
.... x = get_int()
.... print(f"x is {x}")

def get_int():
.... while True:
........ try:
............ x = int(input("What's x? "))
........ except ValueError:
............ print("x isn't int")
........ else:
............ break
.... return x
```

once we get the correct code, we can break.

```
else:
    return x
```

would suffice here.

but... we could also do:

```
def get_int():
.... while True:
........ try:
............ return(int(input(...)))
........ except ValueError:
............ print("x isn't int")
```

↑ could also use "pass" keyword. Simply a keyword that executes nothing.

* Worth noting, the caller of the function has no idea about any errors caught with the try-catch. (that's the point!)

A small improvement, we could instead have the function receive the promp passed by the caller, e.g,

```
def get_int(prompt):
    ....
```