Variables & Data types

dealing with a lot of data. we're going to be working with all types of information, data and values, and sometimes that data can be difficult to manage. So we have this thing called 'variables'
It's basically a container where we can store a certain data value. And when we use a variable if we put those values inside of containers. it makes it a lot easier for us to work with and manage all the different data inside our programs.

A python program

```
print("There once was a man named George, ")
print("he was 70 years old. ")
print("He really liked the name George, ")
print("but didn't like being 70.")
```

Let's say, I wanted to change the character's name. instead of naming character George, I wanted to name the character John.
I'm going to go through and I'm going to have to manually change the name George to the name John at every place inside of this story where it is mentioned.

Now that we changed it, it's going to be updated in our program.

Imagine if I had a story with millions of lines ling. We now have to look through each one of those lines and manually change the character's name. It's not a good way for us to manage the data in our program. So we can use a variable in order to store character's name and character's age and when we use that variable it will make it a lot easier for us to put the character's name and character's age inside of our program.

character_name = "John"
now we have a variable for the character's name
character_age = "35"

```
print("There once was a man named " + character_name + ", ")
print("he was  " + character_age + ",. ")
print("He really liked the name " + character_name + ", ")
print("but didn't like being " + character_age + ",.")
```

When I run this program it's going to print out the same story they were printing out before, but now I don't actually have to type in the character's name and character's age. I can just refer to the variable.
This can be a reallt awesome way to control and manage the data inside of our program.

you can also modify the value. For example, Let's say that halfway through the story, I wanted to change the character's name. I just can make some new lines and assign a new value to one of these variables.

```
print("There once was a man named " + character_name + ", ")
print("he was  " + character_age + ",. ")
character_name = "Mike"
print("He really liked the name " + character_name + ", ")
print("but didn't like being " + character_age + ",.")
```

in the second part of story, it's referring to the name as Mike.

Finally I wanna talk to you guys about the different types of data we can store inside of these variables. A **string** is plain text.

There's also another type of data we can store, which is **numbers**. And when we're storing numbers we don't need these quotation marks. you only need that quotation mark when you're storing a string.

And we also can store what's called a **boolean value**. I can store a value of either True or False.

Working with Strings

in order to create a string, I need to use quotation marks. so I can make open and close quotation marks. Inside of quotation marks I can put whatever text I want the string to have.

One thing I can do would be to create a new line inside of the strings.

```
print("Giraffe\nAcademy")
```

in addition to backslash+n (\n) I could also use a backslash + auotation mark. If I want to put a quotation mark inside of my string I can't just put " because python is going to think that I'm trying to end the string.

\ is called the escape character. Basically tells python that whatever character comes after it we render(?)

```
print("Giraffe\"Academy")
```

you can use backslash to make [1]new lines or [2]prenoun quotation mark or you can use it as [3]a normal backslash.

```
phrase = "Girrafe Academy"
print("Giraffe Academy")
```

concatenation

```
phrase = "Girrafe Academy"
print(phrase + "is cool")
```

functions

we can use functions to modify our strings or get information about our strings.

```
phrase = "Girrafe Academy"
print(phrase,lower()")
```

.lower() / .upper() / .isupper() / print(len(phrase))

.upper().isupper() : I can alse use these functions in combination with each other. You can use these functions one after another.

print(phrase[0]) : G, A string gets indexed starting with zero. if python is counting the characters or it is indexing the characters in a string, it's going to start with zero. 'g' is at position 0 in the string, 'i' is at position 1, etc. We start counting at 0.

print(phrase.index()): tell us where a specific character or string is located inside of our strings

print(phrase.replace("Giraffe", "Elephant"))

there's a lot of these different functions that we can use with strings. If you do a google search you can find all sorts of python functions that you can try out and use. You definitely want to get comfortable working with strings in python because you're gonna be working with them a lot.

Working with Strings

numbers are one of the most common data types in python and any python program you write is most likely going to be dealing with numbers at some point. I want to talk to you guys about the basics of using numbers. We will talk about different types of numbers that we can represent in python. And I also want to show you some awesome functions that we can use with numbers so we can do a certain mathematical operations, mechanism.

print(10 % 3)	my_num = 5 print(str(my_num) + " my favorite number")	my_num = 5 ~~print(my_(num + " my favorite number")~~

my_num = -5 print(abs(my_num))	print(pow(3, 2))	print(max(3, 7, 11)) print(min(3, 7, 11))	print(round(3.2))

from math import * print(floor(3.7))	from math import * print(ceil(3.7))	from math import * print(sqrt(36))	

inside of this math module
when we put this line of code into our program, it gives us access to a lot more math.
you can go online and search for different math functions. There's going to be tons like lists of these things that you can use inside of your program to perform different operations.

Working with Strings

get inputs from users. We're going to allow a user to input information into our program. I'm going to take the information that the user inputs and store it inside of a variable and we're going to be able to do something with that variable.

```
name = input("Enter your name : ")
print("Hello " + name + "!")
```

I'm taking the value that the user inputs and I'm storing it inside of this variable container called name.

Building a Basic Calculator

```
num1 = input("Enter a number : ")
num2 = input("Enter another number : ")
result = num1 + num2
print(result)
```

the result is not quite the answer we were looking for. Because when we get input from a user by default, python is gonna convert it into a string.
I can use a special python function in order to convert those strings into numbers : 1)int, 2)float
float is a number that has decimal.
In certain circumstances you might want the user to only be able to enter into an integer, a whole number. but in a lot of cases, we want them to be able to answer any number that they want so we use float function.

나의 최초 답안

```
~~1st_num~~ = input("Enter a number : ")
~~2nd_num~~ = input("Enter another number : ")
print(1st_num + 2nd_num)
```

```
num1 = input("Enter a number : ")
num2 = input("Enter another number : ")
print(~~value~~(num1) + ~~value~~(num2))
```

List

We can take a bunch of different data values we can put them inside of a list and it allows us to organize them and keep track of them a lot easier. Generally you create a python list and you would put a bunch of related values inside of the list and you can use it throughout your program. We're gonna look at some of the common use case then it really gets you up to speed with what lost are, why they're useful.

We create a list a lot like we create a normal python variable. The first thing you wanna do is giving it a name.
friends = ["Kevin", "Karen", "Jim"]
When we make a list, we're able to store multiple values inside of same object. And then what I can do is I can access these individual items inside of my program.
How can we access individual elements inside of this list? I can actually refer to elements by their index.
The index starts with zero.

print(phrase.index()): tell us where a specific character or string is located inside of our strings

print(friends[0])
we can also access index off of their index from the back of the list : print(friends[-1])
Let's say I wanted to select the elements at index position 1 and all of the elements after that
: print(friends[1:])
print(friends[1:3]) = Karen, Jim

We can also modify a element.
friends = ["Kevin", "Karen", "Jim", "Oscar", "Toby"]
friends[3] = "Mike"

List Functions

extend function : it allows you to take a list and append another list onto the end of it.
luck_numbers = [4, 8, 15, 16, 23, 42]
friends = ["Kevin", "Karen", "Jim", "Oscar", "Toby"]

friends.extend(lucky_numbers)
inside of these parenthese I can pass in the list that I want to add on to the friends list
friends.append("Creed")
we can also add individual elements into a list. the append function always add the item onto the end if the list.
friends.insert(1, "Kelly") / friends.remove("Jim") / friends.clear()
friends.pop() : it basically got rid of the last element inside of the list. It pops the element off of the list.
friends.index("Kevin") :Let's say we wanted to figure out if a certain element was in this list.
friends.count("Jim") : You can also count the number of elements in the list
friends.sort() : We can also sort the list. sort the list in ascending order
friends.reverse() : Reverse the order of the list (sort와 반대가 아님!)
friends2 = friends.copy()

Tuples

Tuple is a type of data structure. It's a container where we can store different values. If you're familiar with lists in python, a tuple is actually very similar to a list. It's basically a structure where we can store multiple pieces of information.

1. How to create a tuple.
One of the most common examples of tuple is coordinates.
coordinates = (4, 5)
one of the things about tuples that makes them unique is a tuple is immutable. The tuple can't be changed or modified. once we create a tuple, you cannot modify it, you cannot change it, you can't add elements to it, you can't erase elements from it, you cna't change any on elements inside the tuple.
print(coordinate[0])
typeError : 'tuple' object does not support item assignment

list vs tuple
If we want to create a list, we would just use square brackets instead of those parentheses. And if I was using a list, I could assign different value to it. I can basically mutate any of the element. I could add or delete, modify, change, whatever I want with the list. But with the tuple, we can't do that.
In practical use cases generally, people use tuple for data that's never going to change.
coordinates = [(4, 5), (6, 7), (80, 34)]
I have a list and inside of it we have these tuple.
리스트 내의 항목을 수정하는 것은 가능한 것으로 확인됨 (eg: (4, 5) → (5, 6))

Functions

If I want to write a function the first thing I have to use is a key word in python. It's called "d e f"
After we type out "def" we need to give this function a name. we need to give this function a name. so just like when we created variables, we give them descriptive name.

```
def say_hi() :
    print("Hello User")

say_hi()
```

- Generally when we're naming functions, you want them to be named in all lower case
- in order to write code that's going to end up being inside the function, we have to indent it. The codes that goes inside of the function need to be indented.
- If I want to execute the code inside of the function, I have to do calling the function. The code is only going to get executed when we specify that we want to execute it.

We can make these functions more powerful and what we can do is we can give them information. A lot of tumes when we write a function we're going to want to have additional information that gets passed in. These are called parameters. A parameter is a piece of information that we give to the function.

```
def say_hi(name, age) :
    print("Hello " + name + ", you are " + age)

say_hi("Mike", "35")
say_hi("Steve", "70")
```

```
def say_hi(name, age) :
    print("Hello " + name + ", you are " + str(age))

say_hi("Mike", 35)
say_hi("Steve", 70)
```

as you go through with python, you are going to be using functions more and more. It's a good idea to break your code up into different functions. So whenever you have a grouping of code designed to perform a specific task, that's a good candidate to be put inside of the function.

Return Statement

Sometimes we want to call a function. We're actually going to want to get information back from the function.

say_hi function에서는 return statement 없이도 결과값을 출력시킬 수 있었는데 무슨 차이가 있는거지?

↳ 별도의 변수를 선언하고 이를 저장하고자 할 때 유용한 듯.
두 번째의 실험에서와 같이 위의 say_hi function과 동일한 출력절차를 이용할 수 있음(function name을 calling).
Building a Basic Calculator에서도 나는 처음에 계산 값을 변수에 저장하는 과정 없이 바로 출력하려고 시도했었음.

```
def cube(num) :
    num*num*num

print(cube(3))
```

```
def cube(num) :
    print(num*num*num)

cube(3)
```

```
def cube(num) :
    return num*num*num

print(cube(3))
```

```
def cube(num) :
    return num*num*num
result = cube(3)
print(result)
```

I'm gonna create a variable called 'result'. This variable 'result' is gonna store the value that gets returned from the cube function. So it's not gonna store cube(3). It's going to store the value that gets returned from executing that function.

I want to point out one more thing. I'm not able to put any code after this return statement. That's because when I use return keyword, it breaks out of the function.

def 안에 print가 포함되어 있으면 해당 함수를 실행했을 때 출력까지 이루어진다.
print 명령을 별도로 실행할 때는 정의된 함수의 결과값이 출력되도록 결과값출력(함수실행) or 함수실행→결과값출력의 명령을 실행한다.

If Statements

```
is_male = True
is_tall = True

if is_male and is_tall :
    print("You are a tall male")
elif is_male and not(is tall) :
    print("You are a short male")
elif not(is_male) and is tall :
    print("You are not a male but are tall")
else :
    print("You are not a male and not tall")
```

```
is_male = True
is_tall = True

if is_male or is_tall :
    print("You are a male or tall or both")
else :
    print("You are neither a male nor tall")
```

```
is_male = True
is_tall = True

if is_male and is_tall :
    print("You are a tall male")
else :
    print("You are either not male or not tall or both")
```

~~if statement의 : 부분 설명있는지 확인하기.~~

If Statements & Comparison

def max_num(num1, num2, num3) :

```
def max_num(num1, num2, num3) :
    if num1 >= num2 and num1 >= num3 :
        return num1
    if num2 >= num1 and num1 >= num3 :
        return num2
    else :
        return num3
print(max_num(3, 4, 5))
```

== : equal, != : not equal to

Building a Better Calculator

The first thing we want to do is getting input from the user. I'm going to create 3 variables. One for the first number, one for the second number and one for the operator.

```
num1 = float(input("Enter first number : "))
op = input("Enter operator : ")
num2 = float(input("Enter second number : "))

if op == "+" :
    print(num1 + num2)
elif op == "-" :
    print(num1 - num2)
elif op == "*" :
    print(num1 * num2)
elif op == "/" :
    print(num1 / num2)
else :
    print("Invalid operator")
```

```
num1 = float(input("Enter first number : "))
op = input("Enter operator : ")
num2 = float(input("Enter second number : "))

if op == "+" :
    result = num1 + num2
elif op == "-" :
    result = num1 - num2
elif op == "*" :
    result = num1 * num2
elif op == "/" :
    result = num1 / num2
else :
    result = "Invalid operator"
print(result)
```

와 같은 구성도 할 수 있을 듯?

I can say float and I can surround this entire input tag with parentheses. And now It's going to imediately convert whatever the user inputs into a float. Now that's gonna mean they're have to insert a number, otherwise we'll get an error. 숫자를 입력해야함? 아니면 숫자로 즉시 변환시키는 거임?(입력 제한이 발생하나? 안하나?)

What we need to do now is we need to figure out what operator the user was trying to do. We can use 'IF' statement in order to figure that out.

Dictionary

Whenever we create a dictionary in python, we're always going to wanna create it inside of these open and close curly brackets. Inside of the dictionary now we can start defining what are called key value pairs. So, I can define a key and that I can give it a corresponding value.

```
monthConversions = {
    "Jan" : "January",
    "Feb" : "February",
    "Mar" : "March",
    "Apr" : "April",
    "May" : "May",
    "Jun" : "June",
    "Jul" : "July",
    "Aug" : "August",
    "Sep" : "September",
    "Oct" : "October",
    "Nov" : "November",
    "Dec" : "December",
}
```

```
monthConversions = {
    0 : "January",
    1 : "February",
    2 : "March",
    3 : "April",
    4 : "May",
    5 : "June",
    6 : "July",
    7 : "August",
    8 : "September",
    9 : "October",
    10 : "November",
    11 : "December",
}
```

기호 [1]{, [2]" ", [3]각 pairs 뒤 ,로 구분

All of these key have to be unique.

Now what we can do is we can access them from inside of this dictionary. So if I wanted to access a specific key or a specific value, all I have to do is referring to the dictionary by name.
Actually there's a bunch of different ways that I can access these month names. I can access different entries inside of this dictionary.

A. The first way is just by making a open and close square bracket.

```
print(monthConversions["Nov"])
```

I'm able to refer to the key, and it's going to the dictionary and it's going to get me the value that's associated to that key

B. get function

```
print(monthConversions.get("Luv","Not a valid key"))
```

What's cool about using this 'get' function is I can specify a default value if the key is not found. There's gonna be certain cases when we're dealing with dictionaries where you're gonna put in a key that might not necessarily map to a value so you put in an invalid key.

While Loop

While Loop is a structure in python which allows us to loop through and execute a block of code multiple times.
I can specify a few different lines of code and then I can put that code inside of a while loop and it will basically loop through that code executing it repeatedly until a certain condition is false.

while loop can be awesome and there's a lot of situations in python we're gonna want to loop through specific lines of code.

```
i = 1
while i <= 10 :
    print(i)
    i += 1
print("Done with loop")
```

```
i = 1
while (condition) i <= 10 :
    print(i)
    i = i + 1
print("Done with loop")
```

① Create an integer(Creating a variable that's a number)
② Create a while loop : while (condition) :
anything that's below the while loop declaration and that's indented is gonna be considered as code that's inside while loop. So that code is going to get repeatedly executed while (the condition) is true.
③ We're gonna keep looping through the code inside the while loop as long as the condition is true.
④ If the condition is not true anymore then we're not going to loop the code anymore then move on.

★ 난이도 上 ★

Building a Guessing Game

나의 초기답안

```
while "Giraffe" = input("Enter your answer :") :
    print("You're right!")

print("Your answer has been submitted")
```

```
answer = input("Enter your answer :") :
while "Giraffe" = answer
    print("You're right!")

print("Your answer has been submitted")
```

위에서의 i <= 10의 조건설정 + while문 내의 증가하는 i 수식 의 조합이 없는
while문의 (조건)이 '문자'로 한 줄로만 설정되어 있으면 적용이 아예 안되는건가? If문을 구지 쓰지 않아도 작동해야 하는거 아닌가 라고 생각되는데.

★ 조건문에 등호는 == 로 쓰여야 한다.
왼쪽 답안은 조건이 일치하는 경우 You're right를 반환하고 다시 input을 요구한다.
오른쪽 답안은 조건이 일치한다면 You're right를 무한 반복하여 반환한다.

정답을 유추하기 위한 문제를 메시지로 미리 제시하는 기능은 없을까?

```
secret_word = "Giraffe"
guess = ""

while secret_word != guess :
    guess = input("Enter guess : ")

print("You win!")
```

If I want to set a limit for the number of times of the users can try to guess the word. Let's say that the user has three tries.

나의 답안

```
secret_word = "Giraffe"
i = 1
guess = ""

while i <= 3 and secret_word != guess :
    guess = input("Enter guess : ")
    i += 1
    print("Try Again!")

print("You win")
```

IF문의 추가가 필요할듯

정답

```
secret_word = "Giraffe"
guess = ""
guess_count = 0
guess_limit = 3
out of guesses = False

while secret_word != guess and not(out_of_guesses) :
    if guess_count < guess_limit :
        guess = input("Enter guess : ")
        guess_count += 1
        print("Try Again!")
    else :
        out_of_guesses = True

if out_of_guesses :
    print("Out of guesses. You lose")
else :
    print("You win")
```

if가 while loop안에서만 쓰였을 뿐만 아니라 결과 메시지를 전달하기 위한 분류의 용도로도 사용되었고 그 과정에서 목적을 완전히 달성하기 위해 out_of_guesses 변수도 생성함.

★ 더 많은 While Loop + IF 문을 경험해 볼 것.
IF문 안의 while loop가능...? 결과 메시지 전달을 위한 별도의 IF문

For Loop

For Loop is a special type of loop in python which allows us to loop over different collections of items. A lot of times we'll use For Loop in python to loop through different arrays, or we can loop over the letters inside of a string or we can loop through a series of numbers. So For Loop provides a very specific purpose.

```
for letter in "Giraffe Academy" :
    print(letter)
```

for + a specified variable,

this variable is going to represent a different value every time we go through this for loop.
This variable is going to be used on every iteration of our for loop, and each time it will most likely have a different value.

for + a specified variable + in + a collection that I wanna loop over

for every letter inside of Giraffe Academy I wanna do something.
we can put what we wanna do with each letter
I can print out this 'letter' variable. And it's going to print out a different letter inside of this Giraffe Academy string on every single iteration of this loop.

여기서 letter은 어떤 성격을 가진 명령어인지 아니면 다른 임의의 글자로 대체가 가능한건지
실행해보니 letter은 대체 가능하고, 다만 아래 설명 중 언급을 보면 단순 string은 아니고 variable로 취급되는 듯?

In addition to using for loop with strings, we can also use for loop with other collections. For example : an array

```
friends = ["Jim", "Karen", "Kevin"]
for name in friends :
    print(name)
```

첫 번째 예시에서는 각 문자단위, 두 번째 예시에서는 각 단어 단위로 어떻게 자동분절이 이루어지는거지?
for loop의 대상이 in뒤에 위치하고 string인 경우 글자, list인 경우 item, range인 경우 각 숫자가 기본단위로 인식된다고 이해하면 될 듯.
단지 variable 설정만을 위한 임의의 문자가 아니라 아래에서는 설정된 variable이 다른 개체의 파라미터로 활용되어 다른 개체와 상호작용이 이루어짐.

```
for index in range(10) :
    print(index)
```

0 ~ 9 까지의 숫자가 하나씩 출력됨

```
for index in range(3, 10) :
    print(index)
```

3 ~ 9까지의 숫자가 하나씩 출력됨

★★★

Inside of the range(), I can pass in the length of the array.

```
friends = ["Jim", "Karen", "Kevin"]
for index in range(len(friends)) :
    print(friends[index])
```

4p List에서의 print(friends[1:3]) = Karen, Jim
for loop이 index를 0, 1, 2를 반환하기 때문에 friends list의 모든 elements가 각각 출력이 이루어짐.

```
friends = ["Jim", "Karen", "Kevin"]
len(friends)
```

if I wanted to get the length of this array, in other words if I wanted to figure out how many elements were inside of it, I can just type out 'len(friends)'

this is gonna spit out 3

변수 index는 in 뒤에서 규정되는 개체 type에 알맞은 값을 가지게 되는 듯. 그리고 그 값이 print문에서 각 사용자가 입력한 code에 따라 활용되는 모습을 보임.

Exponent Function

It allows us to take a certain number and raise it to a specific power.
print(2**3) : This is basically going to be 2 raised to the third power. 2^3.
But I wanna show you guys how we can use for loop in order to create a function something like for loop. So I'll actually create an exponent and I'll use for loop to do it.

```
def raise_to_power(base_num, power_num) :
    result = 1
    for index in range(power_num) :
        result = result * base_num
    return result

print(raise_to_power(3, 5))
```

Let's create a function and now we need to give this function a name. raise_to_power. Inside of here I'm going to accept two parameters. The first parameter is gonna be the base number and the second parameter is gonna be power number.

여기서의 for loop 이용시 생성된 변수인 index는 형식적인 역할 외에 아무런 역할도 안하네? index가 입력된 power_num에 의해 가지게 되는 '값'(0부터 power_num -1까지)들은 raise_to_power함수에 영향을 미치지 않고 값과 무관하게 power_num 만큼으로 **반복되는 횟수**의 형태로만 for loop에 반영되는 듯.

print(base_num**power_num)이 훨씬 쉬운 듯...

```
def raise_to_power(base_num, power_num) :
    return base_num**power_num

print(raise_to_power(2, 5))
```

```
def raise_to_power(base_num, power_num) :
    print(base_num**power_num)

raise_to_power(2, 5)
```

```
def raise_to_power(base_num, power_num) :
    return base_num**power_num

result = raise_to_power(2, 5)
print(result)
```

```
number_grid = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    [0]
]

print(number_grid[2][1])
```

That's basically how we can access elements inside of this 2D list.

for loop inside of for loop. I'll show you how we can use this nested for loop in order to print out all the element inside of this array.
We're gonna create a normal for loop.

```
number_grid = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    [0]
]

for row in number_grid :
    for col in row :
        print(col)
```

Building a Translator

```
def translate(phrase) :
    translation = ""
    for letter in phrase :
        if letter in "AEIOUaeiou" :
            if letter.isupper() :
                translation = translation + "G"
            else :
                translation = translation + "g"
        else :
            translation = translation + letter
    return translation
print(translate(input("Enter a phrase : ")))
```

translate : name of the function / phrase : we want to take a phrase to translate /
The first thing I want to do is create a variable. This would be a final result that we're going to return to the user.
we want to loop through every letter inside of this phrase and if it a vowel, we want to change it into 'g' and if it's not a vowel, we want to leave it alone. As we're looping through phrase and we're going to be adding the letters onto this translation one by one.

Actually there's a special thing we can do in python. We can check to see if something is in something else.

여태까지는 print(translate(phase))와 같이 우리가 상황을 알고있기에 알아서 입력하는 형식을 띄었었는데 이번에는 input구문을 통해 다른 사용자가 입력해야하는 방식을 취해본 듯.

for letter in "AEIOUaeiou" = for letter.lower() in "aeiou"

Comments

A line inside of python file that's just not going to get randered by python. It's going to ignore it. Comments are basically used for us, humans. so a comment is used for me or other developers to write a little comment, plain text inside of a file. #
It's just for me and another developer to come and use. So a lot of times you want to write a little not inside of your python file or if you want to write yourself a little reminder or if you want to write a line to explain a line of code. Now anybody looking at my file would be able to read this and they'll be like "Oh that's what this does. Okay. Cool"
Comments are useful for leaving little comments in python file. If you wanna make comment on multiple lines, you can use a triple quotation marks. '''
Comments can also be useful for doing something called commenting out a line of code. You are not actually deleting the code from your file, you're just commenting it out so python's gonna ignore it.

Try / Except

A lot of thimes when we're writing python programs, you'll encounter different errors. So different situation might come up and your program might throw an error and it might throw an exception. And a lot of times when these situations happen it will completely stop your program from running. What we can do is watch out for a certain specific error that are gonna pop up in our program and we can actually handle them. So instead of our program just breaking and stop executing, we can actually handle those errors and do things when they occur.

```
number = int(input("Enter a number : ")
print(number)
```

I'm prompting the user to enter in a number using this input command and I'm converting whatever they entered into an integer. So as long as they enter in a valid integer I can convert it into a integer. and I can store it inside of this number variable. Let's say I break the rules. I don't enter a number. Ignore what prompt says and I put some random text. **Then program is going to throw an error.**
This is the situation that happens a lot. Up until now we've accepted it as a reality but if you're writing a real live python program, you don't want something like this to trip up your program.

We can use something called try except block. Depending on what happens, we can do different things. If the user enters in something wrong, it's gonna be able to catch it

```
value = 10/0
try :
    number = int(input("Enter a number : ")
    print(number)
except :
    print("Invalid Input")
```

ZeroDivisionError : division by zero

```
try :
    value = 10/0
    number = int(input("Enter a number : ")
    print(number)
except :
    print("Invalid Input")
```

Invalid Input

we didn't input something that was invalid but it wasn't able to handled and it threw an error. This brings me up to another point. We can actually catch or accept specific type of errors.

```
try :
    number = int(input("Enter a number : ")
    print(number)
except ZeroDivisionError :
    print("Divided by zero")
except ValueError :
    print("Invalid Error")
```

```
try :
    answer = 10/0
    number = int(input("Enter a number : ")
    print(number)
except ZerodivisionError as err:
    print(err)
except ValueError :
    print("Invalid Input")
```

에러 유형의 자동완성목록이 나타나는 것을 보니 성질을 갖는 몇몇 타입이 이미 정해져 있는 듯.
And I want to show you guys one more thing. We can actually store this error as a variable.

10/0을 input값에 넣으면 ValueError메세지가 출력됨...

```
def say_hi() :
    print("Hello User")

say_hi()
```

```
def raise_to_power(base_num, power_num) :
result = 1
for index in range(power_num) :
result = result * base_num
print(result)
```