Getting Started With Python Programming: Part II

- Converting between different types of data
- Operator overloading
- Formatting text output
- Different types of programming errors

Storing Information: Bottom Line (You Need Know)

- Information is stored differently between strings, integers and floats
  - E.g. *String* “1” = `00110001` (ASCII) whereas the *number* 1 = `1111 1111` (2s complement).
- For now: you don’t have to know how these values were obtained.
- Why it important to know that different types of information is stored differently?
  - One motivation: sometimes students don’t why it’s significant that “123” is not the same as the number 123.
  - Certain operations only apply to certain types of information and can produce errors or unexpected results when applied to other types of information.
- **Example**

```
num = input("Enter a number")
numHalved = num / 2
```

Program ‘crashes’: Cannot perform division operation on a string

James Tam

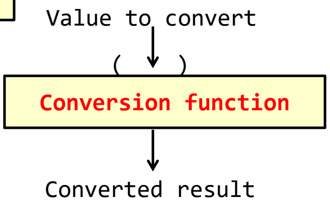
Converting Between Different Types Of Information

- Example motivation: you get user input in the form of a string but also you need to perform calculations on that input.
- Some of the conversion mechanisms (functions) available in Python:

Format:

```
int(<value to convert>)
float(<value to convert>)
str(<value to convert>)
```

Digits right of decimal are removed (truncation - no rounding)



Examples:

Name of the full example: `1convert.py`

```
var1 = 10.9
var2 = int(var1)
print(var1,var2)
```

```
10.9 10
```

- An error occurs if the conversion cannot be made e.g. `int("t")`

James Tam

Overloaded Operators

- The same symbol can have different results depending upon the context.
- One example of where this issue may come up is when you don't convert the variable type when you should e.g. you get user input in the form of a string but you don't convert it to a number prior to performing a mathematical operation.
- Example: the 'plus' operator **+**
  - Previously this symbol represented mathematical **addition** because the values left and right of the symbol (operands) were numeric e.g.,

    ```
    num1 = 2 + 2
    ```
  - If the operands are strings then the symbol represents the string operation **concatenation** e.g.,

    ```
    str1 = "2" + "2"
    ```

James Tam

Overloaded Operators (2)

- **Name of the full example:** `2overloadedOperator.py`

```
var1 = "100"
var2 = "-10.5"
# Concatenation operation (combines two strings).
print(var1 + var2)

# Addition operation is performed.
print(int(var1) + float (var2))

#Error cannot perform a concatenation on a number
str2 = "2" + 2
```

James Tam

Converting Types: Extra Practice For Students

- Determine the output of the following program:

```
print(12+33)
print("12"+"33")
x = 12
y = 21
print(x+y)
print(str(x)+str(y))
a = 2 * 0.5
print(a)
print(3/0.5)
```

To determine the result of dividing by a rational number, follow these steps:

1. Express the rational number in fraction form if necessary.
2. Multiply the first rational number by the reciprocal of the second rational number (the divisor).
3. Multiply the numerators together and the denominators together to find the result.

Bing generated synopsis
- That is multiply by the reciprocal of the rational value.

James Tam

More On Getting User Input

- **Format:**

```
<variable / memory location> = <name of the function i.e.
input>(<Optional: a string that acts as the prompt>)
```

- **Example:**

```
lastName = input("Family (last) name: ")
```

- Python 3.x: the value returned by input is a string

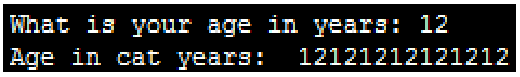
James Tam

Converting Between Different Types Of Information: Getting Numeric Input

- Since the ‘`input()`’ function <u>only returns a string</u> so the value returned must be converted to the appropriate type as needed.
  - **Name of the full example:** `3convert4Input.py`

```
# No conversion performed: problem!
HUMAN_CAT_AGE_RATIO = 7
age = input("What is your age in years: ")
catAge = age * HUMAN_CAT_AGE_RATIO
print ("Age in cat years: ", catAge)
```

- **‘Age’ refers to a string not a number.**
- **The ‘*’ is not mathematical multiplication (repetition operator)**



```
What is your age in years: 12
Age in cat years:  12121212121212
```

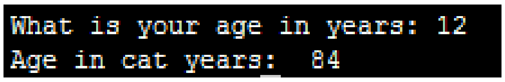
James Tam

Converting Between Different Types Of Information: Getting Numeric Input (2)

```
# Input converted: Problem solved!
HUMAN_CAT_AGE_RATIO = 7
ageString = input("What is your age in years: ")
ageNum = int(ageString)
catAge = ageNum * HUMAN_CAT_AGE_RATIO
print("Age in cat years: ", catAge)


print("Alternative: combines 2 steps into 1")
age = int(input("What is your age in years: "))
catAge = age * HUMAN_CAT_AGE_RATIO
print("Age in cat years: ", catAge)
```

- **'Age' converted to an integer.**
- **The '*' now multiplies a numeric value.**



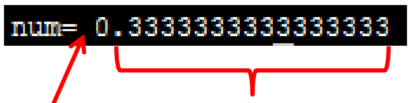
```
What is your age in years: 12
Age in cat years:  84
```

James Tam

By Default Output Is Unformatted

- Example:

```
num = 1/3
print("num=",num)
```



```
num= 0.3333333333333333
```

Sometimes you get extra spaces (or blank lines)

The number of places of precision is determined by the language not the programmer

- There may be other issues e.g., you want to display output in columns of fixed width, or right/left aligned output
- There may be times that specific precision is needed in the displaying of floating point values

James Tam

Formatting Output

- **Original approach:** but compatible with many languages such as C (`printf`), Java (`System.out.printf`) (covered if there is time): **format specifiers** and **escape codes.**
- **Second approach developed:** using the `Format` class (Java has an equivalent class `MessageFormat`).
- **Current approach for python:** using f-string.
  - Powerful (most options)
  - Some find it more complicated than the original approach.
  - As of 2025: You may note fewer resources available that clearly AND completely explain its usage (typically you see examples for a few specific cases).
  - No wide spread equivalents in other languages e.g. in Java you can try ~`format()` method of class `String`.

James Tam

Format Specifiers (If There's Time)

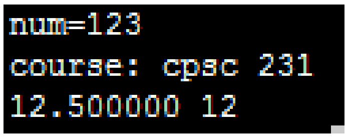
- **Format**:

```
print ("%<placeholder for type of info to display/code>"
  %<source of the info to display>)
```

Doesn't literally display this: It's a placeholder (for information to be displayed)

- **Example (starting with simple cases)**:
  - **Name of the full example:** `4formatSpecifiersAsPlaceholders`

```
num = 123
st = "cpsc 231"
print("num=%d"        %num)
print("course: %s"    %st)
num = 12.5
print("%f %d" %(num,num))
```



```
num=123
course: cpsc 231
12.500000 12
```

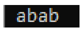
James Tam

Types Of Information That Can Be Formatted Via Format Specifiers (Placeholders: If There's Time)

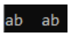
Specifier	Type of Information to display
%s	String
%d	Integer
%f	Floating point
%g	"Scientific notation" (similar to floating point in terms of output)

James Tam

Format Specifiers: Precision & Field Width (If There's Time)

- **Precision**:
  - The number of digits to the right of the decimal point.
    - E.g. 3.14 has 2 places of precision
  - Alternate ways of specifying this term as: number of places of precision, number of fractional digits
- **Field width**:
  - Think of it as "the width of a column" (the column created for each format specifier/placeholder).
    - E.g. 1: Four column width %4s
    - E.g. 2: Ten column width %10d
  - When the column is too narrow to display the data then the column width is automatically expanded.
  - When the column is wider than the width of the data then extra spaces will be added before or after the data.
    - Space before the first "ab" and a space after the second "ab" 

```
 abab 
```

    - Space after the first "ab" and a space before the second "ab" 

```
ab  ab
```

James Tam

Formatting Effects Using Format Specifiers (If There's Time)

- **Format**:

  `%<field width>[1].<precision>[2]<type of information>`

- **Examples (format specifiers to format output)**:
  - **Name of the full example**: `5formatSpecifier4AlignmentNPrecision`

```
num1 = 12.55
num2 = 12
str1 = "hi"
print ("%s" %str1)                hi
print ("%3.1f" %num1)             12.6      1 2 . 6
print ("%6.1f" %num1)               12.6    <SP><SP> 1 2 . 6
print("%-5s" %num2)               12        1 2 <SP><SP><SP>
print ("%3s%-3s" %("ab", "ab"))    abab     <SP> a b a b <SP>
print ("%-3s%3s" %("ab", "ab"))   ab  ab    a b <SP> <SP> a b
```

- 1 A positive integer will add leading spaces before the information to display (right align), negatives will add trailing spaces (left align). Excluding a value will set the field width to a value large enough to display the output
- 2 For numeric variables only.

James Tam

Escape Codes/Characters (If There's Time)

- The back-slash character enclosed within quotes won't be displayed but instead indicates that a formatting (escape) code will follow the slash.

Escape sequence	Description
`\a`	Alarm: Causes the program to beep.
`\n`	Newline: Moves the cursor to beginning of the next line.
`\t`	Tab: Moves the cursor forward one tab stop.
`\"`	Double quote: Prints a double quote.
`\\`	Backslash: Prints one backslash.

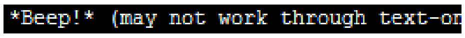
  - Escape codes can be used in 'C' and 'Java (same usage as well).

James Tam

Escape Codes (2: If There's Time)

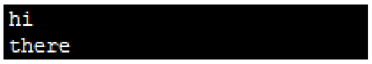
- **Program name:** `6escapeCodes.py`

```
print ("\a*Beep!*")
```



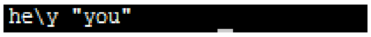
```
*Beep!* (may not work through text-on
```

```
print ("hi\nthere")
```



```
hi
there
```

```
print ("he\\y \"you\"")
```



```
he\y "you"
```

James Tam

Text Formatting: F-String

- The newest of the three approaches with the most options but it is specific to python.
  - Not a function such as: `print()` or `input()`
  - Nor is it an operator such as: `+`, `-`, `*`, `/`.
  - It is a feature of python syntax that allows strings to be formatting according to the expression.
- **General format:**

```
print(f"<string to be formatted")
```

- **Example:**

```
print(f"4 learning only: 4 string without formatting")
```

James Tam

Basic Use Of F-String

- **Name of the full example:** `7f_string_basic.py`

```
num1 = 1/8
precision = -1
```

```
Unformatted: f-string not used num= 0.125
```

```
print("Unformatted: f-string not used num=",num1)
```

```
Unformatted: f-string used num=0.125
```

```
print(f"Unformatted: f-string used num={num1}")
```

```
3 places of precision num=0.125
```

```
print(f"3 places of precision num={num1:0.3}")
```

```
Number of places of precision (0+): 2
```

```
precision = int(input("Number of places of precision (0+): "))
```

```
2 rational digits num=0.12
```

```
print(f"{precision} rational digits num={num1:0.{precision}}")
```

James Tam

Aligning Output: F-String

- **Format:**

```
print(f"{<display data>:<field width>}") #L-align: trailing spaces
print(f"{<display data>: > <field width>}") #R-align: leading spaces
```

- **Examples:**

```
#Valid but not mandatory
print(f"Name: {name:<7}is me.")
```

```
print(f"Name: {name:7}is me.") #L-align: trailing spaces
print(f"Age={age:>3}") #R-align: leading spaces
```

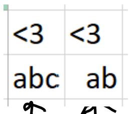
- **Name of the full example:** `8f_string_alignment.py`

```
name = "JAMES"
age=37
print(f"Name:{name:7}is me.")
```

```
JAMES (needs 5 'slots')
Name:J A M E S SP SP is me.
```

James Tam

Aligning Output: F-String (2)



<3 <3
abc ab
Column width matches width of data
Column winder than the width of data (right-align: leading 'space')

#No greater than, equal to Co-pilot says "3 or greater"
print(f"Age={age:>=3}")

```
age=37
print(f"Age={age:>1}")        3 7        37 (needs 2 'slots')
```
Field width is 1

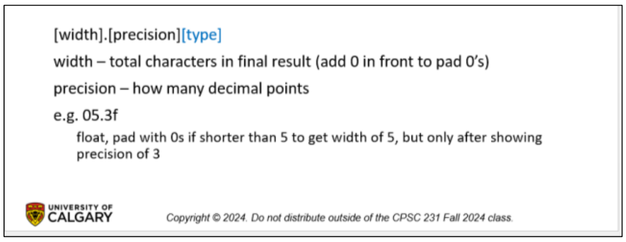
Field width is 2
```
print(f"Age={age:>2}")        3 7        37 (needs 2 'slots')
```

Field width is 3
```
print(f"Age={age:>3}")        SP 3 7     37 (needs 3 'slots')
```

James Tam

F-String Aligning Output/Precision: "Required Knowledge"

- FYI: This is how this topic was presented in the notes provided by the course coordinator (Dr. Michelle Cheatham)



[width].[precision][type]

width – total characters in final result (add 0 in front to pad 0's)

precision – how many decimal points

e.g. 05.3f

float, pad with 0s if shorter than 5 to get width of 5, but only after showing precision of 3

UNIVERSITY OF CALGARY

James Tam

Types Of Information: F-String

- **Types:**

Type	Information displayed	Symbol
Integer	Whole numbers	d
Float	~Rational numbers	f
String	Characters	s
Scientific notation	S.N.: displays as an exponent	e
General	Scientific notation (depends upon # rational digits)	g

- **Format** (to specify the **type** of the **information**)

```
print(f"…{<information>¹:<type>} ")
```

- **Some examples:**

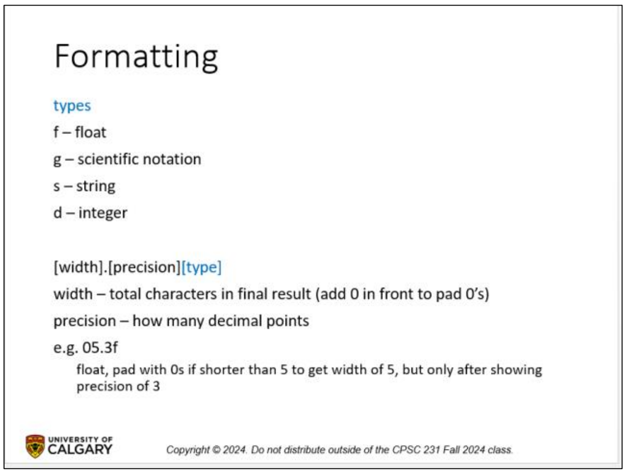
```
print(f"Display as integer{num2:4d}")
print(f"Display in scientific notation {1/3:e}")
```

1: Information can take the form of a variable (name), named constant (PI) or an unnamed constant (e.g. 3.14, "TAX_RATE")

James Tam

F-String Data Types: "Required Knowledge"

- FYI: This is how this topic was presented in the notes provided by the course coordinator (Dr. Michelle Cheatham)



Formatting

types

f – float

g – scientific notation

s – string

d – integer

[width].[precision][type]

width – total characters in final result (add 0 in front to pad 0's)

precision – how many decimal points

e.g. 05.3f

float, pad with 0s if shorter than 5 to get width of 5, but only after showing precision of 3

UNIVERSITY OF CALGARY

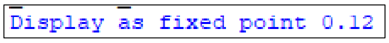
Copyright © 2024. Do not distribute outside of the CPSC 231 Fall 2024 class.

James Tam

Types Of Information: F-String

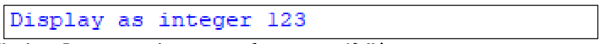
- **Name of the full example:** `9f_string_display_types.p`

```
num1 = 1/8
num2 = 123
string1 = "abc"
```



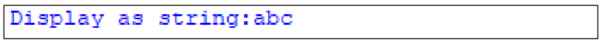
```
Display as fixed point 0.12
```

```
print(f"Display as fixed point {num1:0.2f}")
```



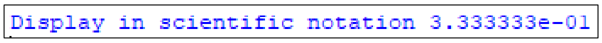
```
Display as integer 123
```

```
print(f"Display as integer{num2:4d}")
```



```
Display as string:abc
```

```
print(f"Display as string:{string1:s}")
```



```
Display in scientific notation 3.333333e-01
```

```
print(f"Display in scientific notation {1/3:e}")
```

James Tam

Why Bother Specifying The Type

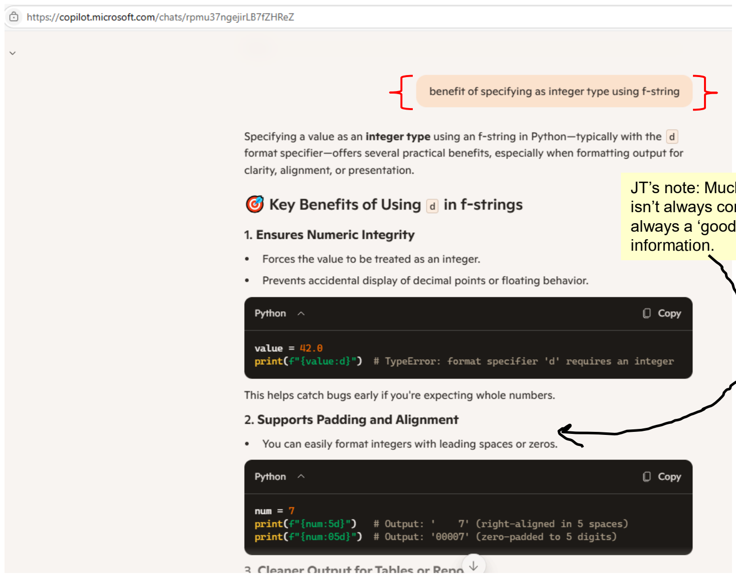
- That is, the previous examples would have ‘worked’ without using type specifies (e.g. ‘f’ for “floating point”)
- You can read discussions online but here’s one quick reason:
  - type checking:
    - If information can only be of a certain ‘type’ then the program can flag the incorrect type as a visible error rather than producing a bug in the program.
    - Example: if you only want an ID number to consist only of digits then specify the type of display as integer, something such as a string will produce an error.

```
num1 = 1/8
num2 = 123
```

```
print(f"{num1:s}") #can’t display a float as a string
print(f"{num2:s}") #can’t display an int as a string
```

James Tam

Example: More Information On Types, **How AI Can Be Legitimately Used (Learning)**


https://copilot.microsoft.com/chats/rpmu37ngejirLB7fZHReZ

benefit of specifying as integer type using f-string

Specifying a value as an **integer type** using an f-string in Python—typically with the `d` format specifier—offers several practical benefits, especially when formatting output for clarity, alignment, or presentation.

🎯 Key Benefits of Using `d` in f-strings

1. **Ensures Numeric Integrity**

- Forces the value to be treated as an integer.
- Prevents accidental display of decimal points or floating behavior.

```python
value = 42.0
print(f"{value:d}")  # TypeError: format specifier 'd' requires an integer
```

This helps catch bugs early if you're expecting whole numbers.

2. **Supports Padding and Alignment**

- You can easily format integers with leading spaces or zeros.

```python
num = 7
print(f"{num:5d}")   # Output: '    7' (right-aligned in 5 spaces)
print(f"{num:05d}")  # Output: '00007' (zero-padded to 5 digits)
```

3. Cleaner Output for Tables or Repo

JT's note: Much like the web, AI isn't always correct nor is it always a 'good' source of information.

Source: Bing Co-pilot

James Tam

Formatting Text Output Vs. Changing Variables

- Features such as: F-String for specifying precision and alignment and format specifiers only affect the information displayed for functions such as print.
- **They DO NOT change the value stored!**
- Example:

```
num = 1/3
```

```
Num's displayed value here=0.3
```

```
print(f"Num's displayed value here={num:3.1}")
```

```
Num's actual stored value=0.3333333333333333
```

```
print(f"Num's actual stored value={num}")
```

James Tam

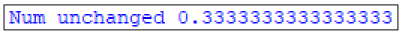
Functions *Can* Change Values In Variables

- Recall: variables are changed via an **assignment**
  - Example:

```
num = 12
print(num)  #Num passed to a function not changed.
num = 21    #Updates value stored in num.
```

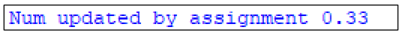
- Some functions (such as 'round' can return a modified value of what was passed into it).
  - Example:

```
num = 1/3
round(num,2)
print("Num unchanged",num)
```



```
Num unchanged 0.3333333333333333
```

```
num = round(num,2)
print("Num updated by assignment",num)
```



```
Num updated by assignment 0.33
```

James Tam

F-String: Easy Pitfall (To 'Fall' Into)

- Tam says: "**Small details matter** because they can produce drastic effects, pay attention to syntax (e.g. the 'Format' headings in my notes)!"
- This isn't a "Tam A.R. thing" it's a property of all programming languages.
- Example what you if exclude the '**f**' in an attempt to use F-String.

```
num = 1/3
print("{num:3}")  #Incorrect: "literal string" between quotes
print(f"{num:3}") #Correct: 'f': specifies the use of F-String
```

James Tam

Types Of Programming Errors

1. Syntax/translation errors
2. Runtime errors
3. Logic errors

James Tam

1. Syntax/ Translation Errors

- Each language has rules about how statements are to be structured.
- An English sentence is structured by the *grammar* of the English language:
  - My cat sleeps the sofa.

Grammatically incorrect (FYI: missing the preposition to introduce the prepositional phrase 'the sofa')

- Python statements are structured by the *syntax* of Python:

```
5 = num
```

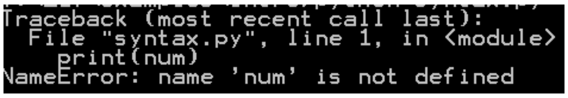
Syntactically incorrect: the left hand side of an assignment statement cannot be a literal (unnamed) constant (or variable names cannot begin with a number)

James Tam

1. Some Common **Syntax Errors**

- Miss-spelling names of keywords
  - e.g., '`primt()`' instead of '`print()`'
- Forgetting to match closing quotes or brackets to opening quotes or brackets e.g., **print("hello)**
- Using variables before they've been named (allocated in memory).
- **Name of the full example**: `10error_syntax.py`

```
print(num)
num = 123
```

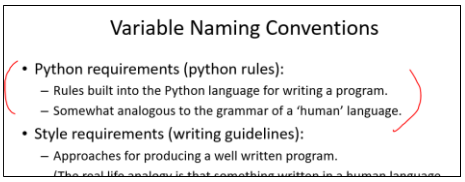


```
Traceback (most recent call last):
  File "syntax.py", line 1, in <module>
    print(num)
NameError: name 'num' is not defined
```

James Tam

1. Syntax Errors: Rules For Specifying Python Instructions

- The rules were introduced in the previous section:



Variable Naming Conventions

- Python requirements (python rules):
  - Rules built into the Python language for writing a program.
  - Somewhat analogous to the grammar of a 'human' language.
- Style requirements (writing guidelines):
  - Approaches for producing a well written program.
  - (The real life analogy is that something written in a human language

- The python 'rules' are specified in the syntax of the language.

James Tam

2. Runtime Errors

"My computer crashed!"

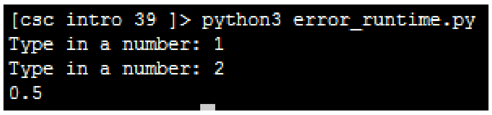
- Occur as a program is executing (running).
- The syntax of the language has *not* been violated (each statement follows the rules/syntax).
- During execution a serious error is encountered that causes the execution (running) of the program to cease.
- A common example of a runtime error is a division by zero error.
  – Another example is a type error e.g. `var = "1" + 1`
  – We will talk about other run time errors later.

James Tam

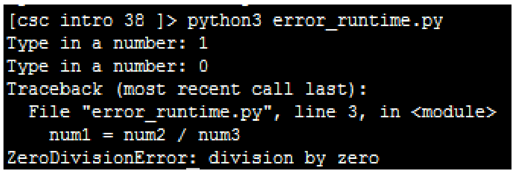
2. Runtime Error[1]: An Example

- **Name of the full example**: `11error_runtime.py`

```
num2 = int(input("Type in a number: "))
num3 = int(input("Type in a number: "))
num1 = num2 / num3 # When zero is entered
print(num1)
```



```
[csc intro 39 ]> python3 error_runtime.py
Type in a number: 1
Type in a number: 2
0.5
```



```
[csc intro 38 ]> python3 error_runtime.py
Type in a number: 1
Type in a number: 0
Traceback (most recent call last):
  File "error_runtime.py", line 3, in <module>
    num1 = num2 / num3
ZeroDivisionError: division by zero
```

1 When 'num3' contains zero

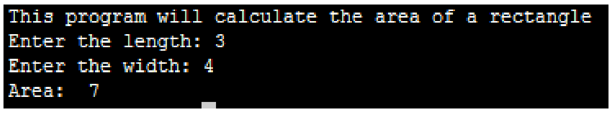
James Tam

3. Logic Errors

Software "bugs"

- The program has no *syntax errors*.
- The program runs from beginning to end with *no runtime errors*.
- But the logic of the program is incorrect (it doesn't do what it's supposed to and may produce an incorrect result).
- **Name of the full example**: `12error_logic.py`

```
print ("This program will double the number.")
number = int(input("Type in the number to be doubled: "))
doubledIt = number + 2
print("Number: %d, Doubled: %d" %(number,doubledIt))
```



```
This program will calculate the area of a rectangle
Enter the length: 3
Enter the width: 4
Area:  7
```

James Tam