

2. Data in Program

Victor Lee

Department of Electrical and Electronic Engineering



Program needs Data

Data

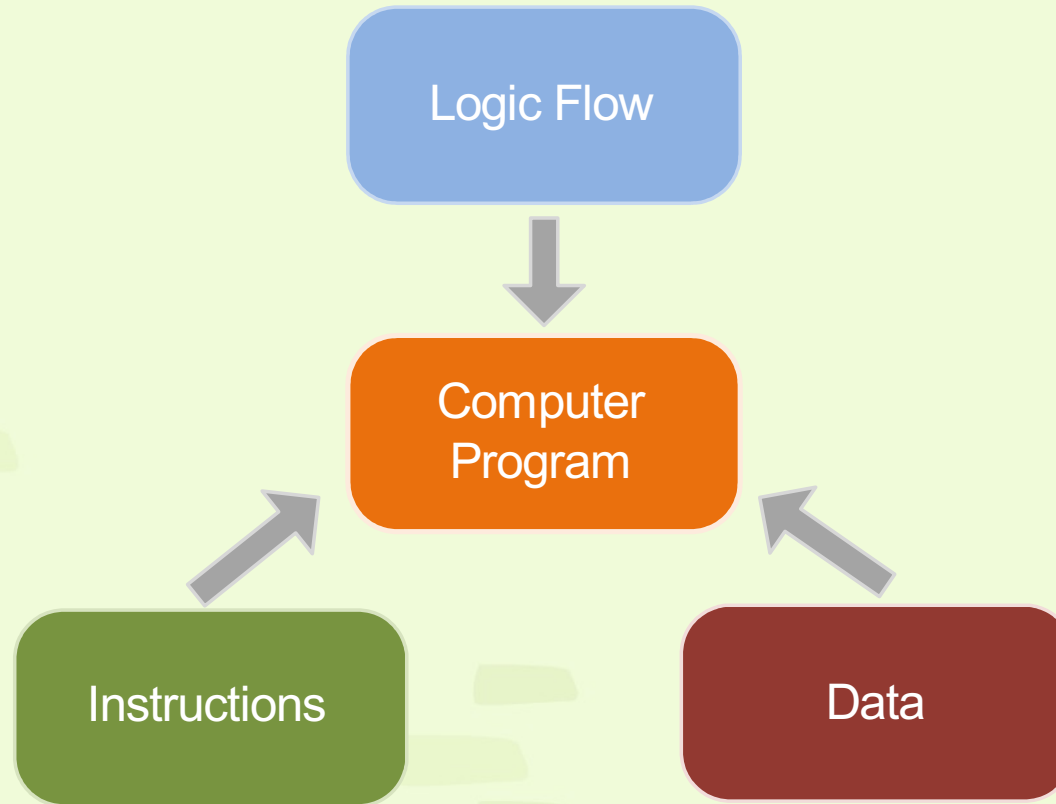
```
weight=70 #weight in kg  
height=1.65 #height in m  
bmi=weight/height/height  
print(bmi)
```

Data

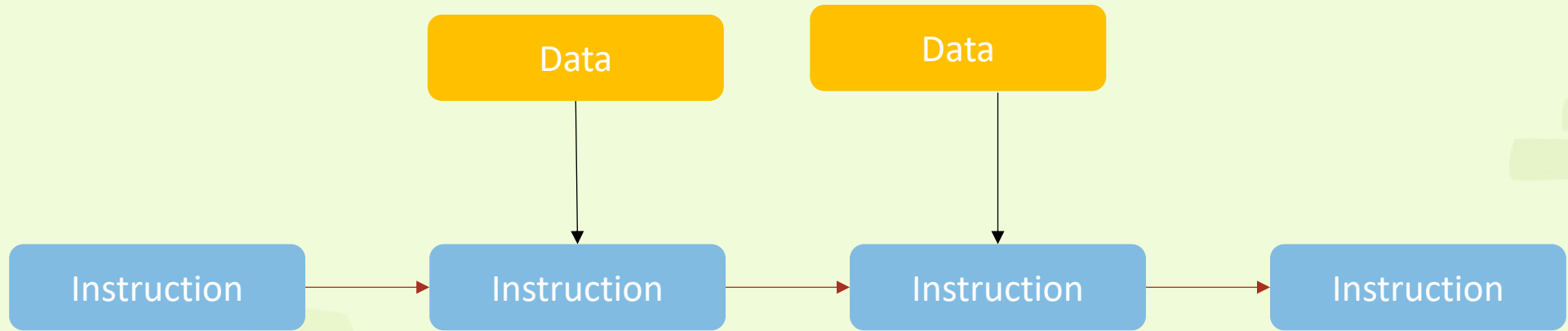
Data



Computer Program



Computer Program = Instruction + Logic Flow + Data Access



Literal

- In programming, literal represents a fixed data value (immutable)
- Literal in Python include numbers and strings

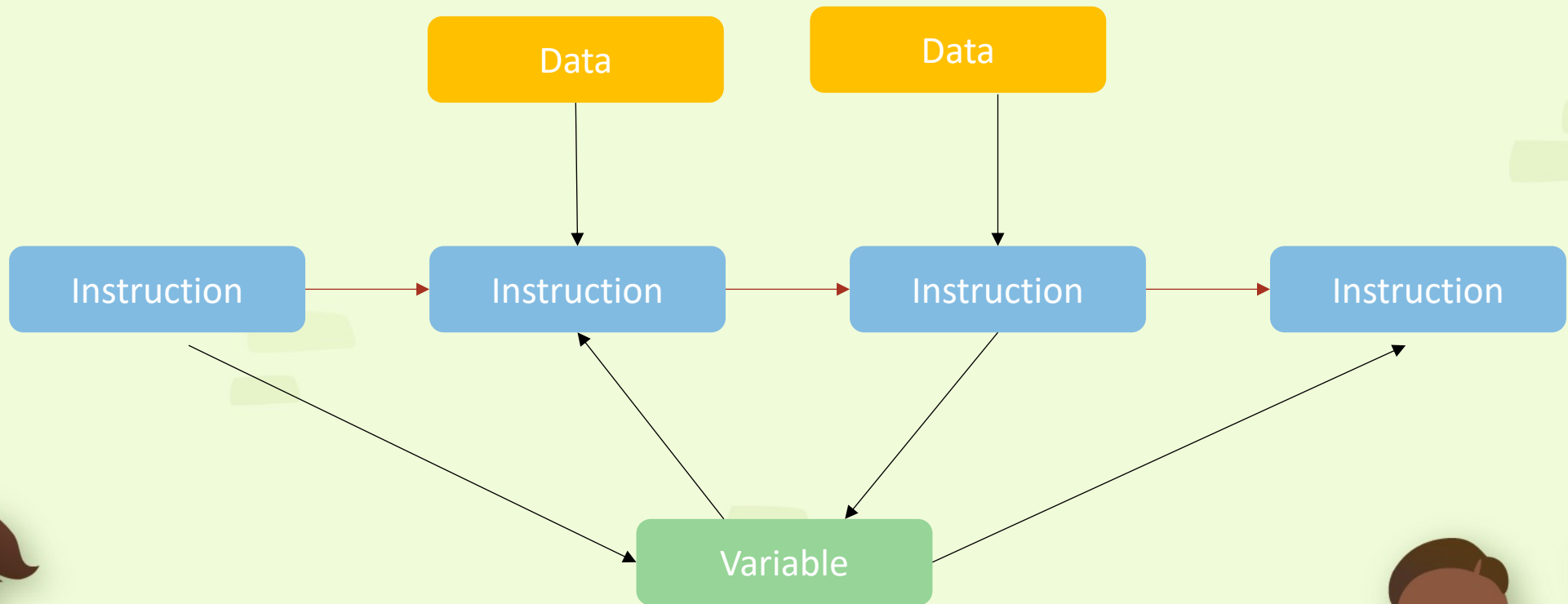
Literal

```
weight=70 #weight in kg  
height=1.65 #height in m  
bmi=weight/height/height  
print(bmi)
```

Literal



What if the data have to be changed during execution?



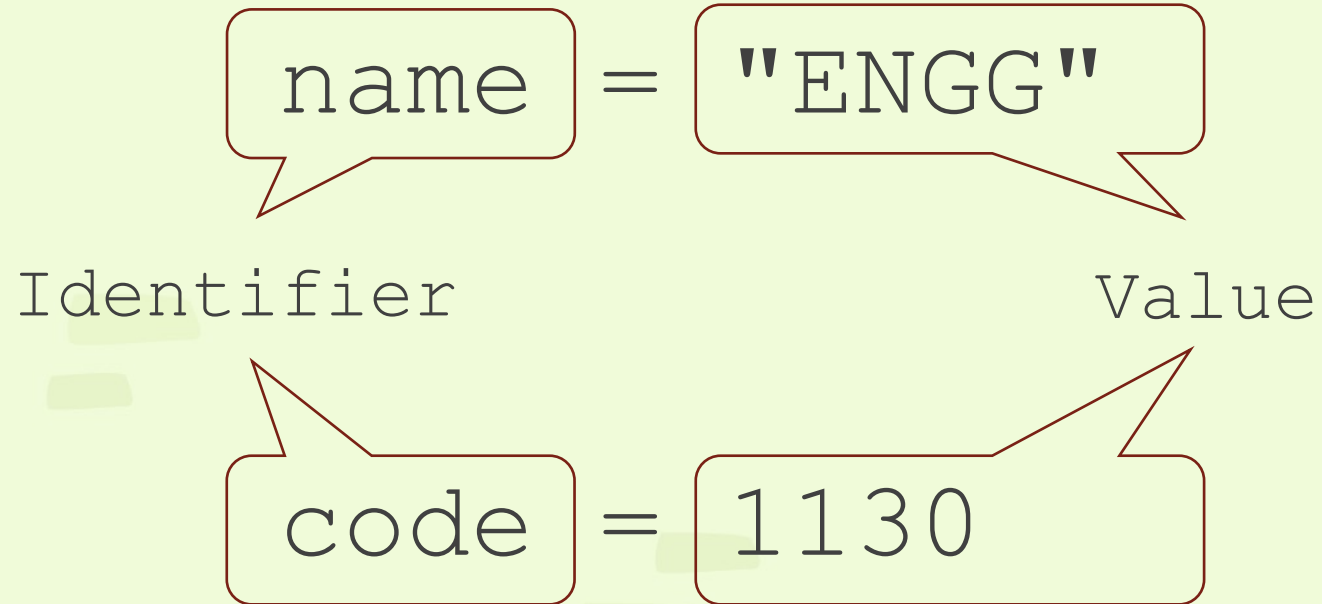
Variable

A space for programmer to store a temporary value for later use



Variable in Python: Syntax

- Create a new variable and give it a value in an assignment statement



Variable

Variable

```
weight=70 #weight in kg  
height=1.65 #height in m  
bmi=weight/height/height  
print(bmi)
```

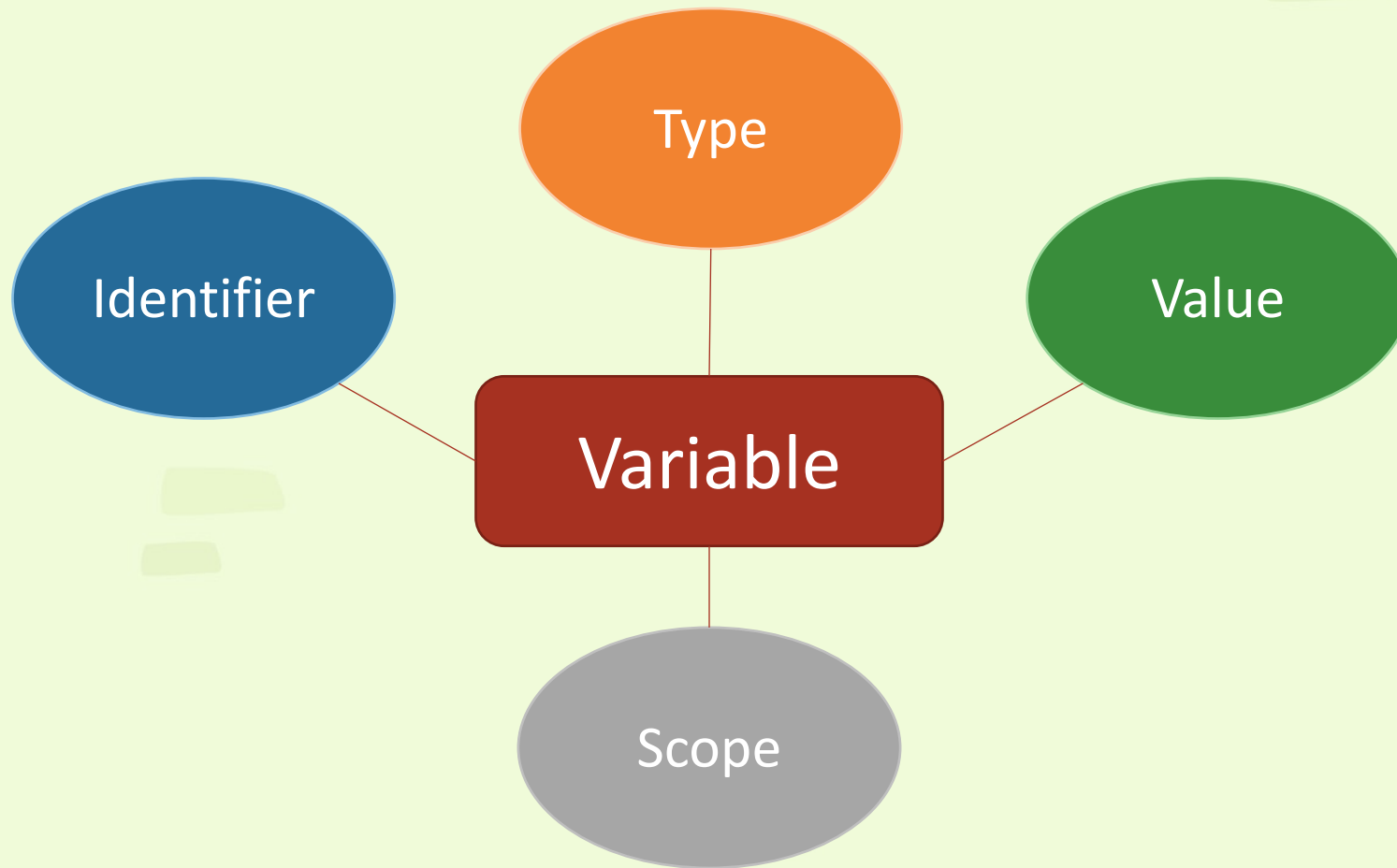
Expression:
Combination of literals,
variables and operators

Variable

Variable



Basic Concepts of Variable



Variable's Attributes

- Identifier (Locker #)
 - Name of a variable used to uniquely identify the variable
- Type (Locker type)
 - The type of a variable, i.e., what kind of data it supposes to store
- Value (Locker content)
 - The value (data) stored in a variable
- Scope
 - Will be covered later



Rules for Identifier

- Length: Unlimited
- Can have a mix of letters and numbers, however,
 - cannot start with number
 - case sensitive

```
IDLE Shell 3.11.1
>>> code=1330
>>> print(Code)
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    print(Code)
NameError: name 'Code' is not defined. Did you mean: 'code'?
```

- Underscore ''': OK (often used to separate words in a long name)
- No Python keywords (reserved words)



Identifier should be

- No duplication (unique)
- Easy to understand (meaningful: describe what the variable is used for)
 - x01, n2, n3 vs point_X, total_count, record_count
- Easy to remember
 - n_of_record vs no_of_record vs record_count
- Not confusing
 - NO1 vs N01



Python's 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

```
IDLE Shell 3.11.1
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await',
'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is',
'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try',
, 'while', 'with', 'yield']
```

Ln: 24 Col: 295



Data Type in Python

- Integer (int)
- Floating-point number (float)
- Complex number (complex)
- String (str)

The data type can be checked using the built-in function – `type`.

```
IDLE Shell 3.10.0
>>> type(1330)
<class 'int'>
>>> type('ENGG1330')
<class 'str'>
>>> type('GPA: ')
<class 'str'>
>>> type(3.8)
<class 'float'>
>>> type(1+2j)
<class 'complex'>
>>> |
Ln: 97 Col: 0
```



Integer

- zero, positive and negative whole number
- 1, 2, 3, -1, 2, 0
- In computer, a fixed number of bits are used to store / represent an integer

```
retryCount = 3
```

```
playerID = 10120
```



How many numbers can be represented with 1 bit?

0
1

0	0	0
0	1	1
1	0	2
1	1	3

--	--	--

2^n



Some Facts about Integer

- In Python, integers have unlimited size.
- Integers can be binary, octal, and hexadecimal values.
- Binary starts with a leading zero and b (0b or 0B), e.g., 0b10100110010 (1330 in decimal)
- Octal starts with a leading 0o or 0O, e.g., 0o2462 (1330 in decimal)
- Hexadecimal starts with a leading 0x or 0X, e.g., 0x532 (1330 in decimal)



Float

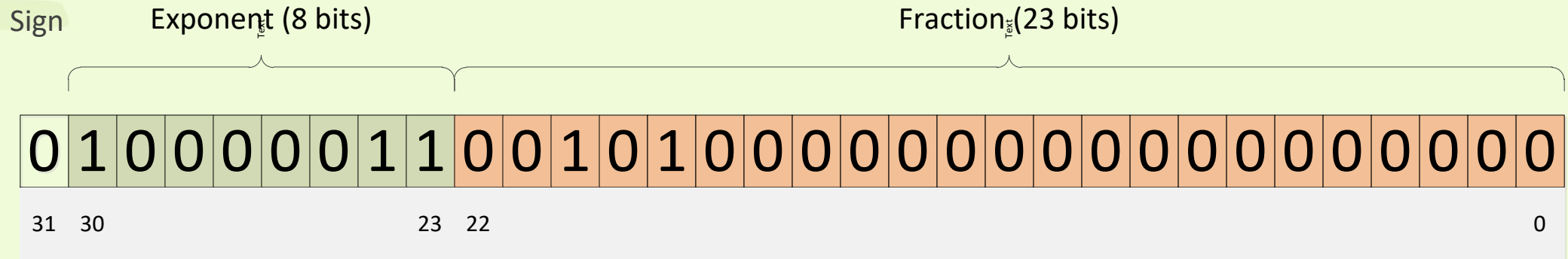
- Floating-point numbers are real numbers with a decimal point and fractional part
- -1.2, 3.0, 4.2,-9.423
- In computer, a fixed number of bits are used to represent a floating-point number

```
pi = 3.1415
```

```
price = 2.3
```



How floating-point number is stored in a computer?



131
$2^{(131-127)}$

$1/2^3$
0.125

$1/2^5$
0.03125

$$2^4=16$$

X

$$1 + 0.125 + 0.03125 = 1.15625$$

=18.5



How floating-point number is stored in a computer?

- The more bits are used to store the fraction, the more precise the floating-point number will be.
- Python uses 64 bits to represent floating-point numbers.

Type	Size	Fraction size	Possible combination
float	32 bits	23 bits	$\frac{1}{2}, \frac{1}{2^2}, \frac{1}{2^3}, \dots, \frac{1}{2^{23}}$
double	64 bits	52 bits	$\frac{1}{2}, \frac{1}{2^2}, \frac{1}{2^3}, \dots, \frac{1}{2^{23}}, \dots, \frac{1}{2^{52}}$



Complex numbers

- Consist of real and imaginary parts (adding a j or J suffix)

$$a = 2 + 1j$$

$$b = 4 + 2j$$

$$c = a + b$$



Syntax

Variable = value

e.g., $x = 5$

Literals: 3, 4

Variable: a, y

Expression: $a + 3$

Expression: Combination of
literals, variables and operators



Look out!

- Python is a weakly type language that variable can be changed to any unrelated type during run-time
- It is programmer's responsibility to keep track of variable type and prevent invalid expression

```
IDLE Shell 3.11.1
>>> n=2
>>> m=1.4
>>> n=m+n
>>> n="1"
>>> m=n+5
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    m=n+5
TypeError: can only concatenate str (not "int") to str
Ln: 110 Col: 0
```



Operators

- Used to perform operations on literals and variables.
- Common groups of operators
 - Arithmetic operators
 - Assignment operators
 - String operators



Arithmetic Operators

Operator	Description	Example	Output
+	Addition	$3 + 4$	7
-	Subtraction	$1 - 4$	-3
*	Multiplication	$4 * 5$	20
/	Division	$5/2$	2.5
**	Exponentiation	$3**2$	9
%	Modulus: divides two numbers and returns the remainder	$5\%2$	1
//	Floor division: divides two numbers and rounds down to an integer	$5//2$	2

Operand is the value on which an operator operates.



Operation Order: PEMDAS

- **P**arentheses: ()
- **E**xponentiation: **
- **M**ultiplication and **D**ivision: *, /
- **A**ddition and **S**ubtraction: +, -
- Operators with the same precedence such as * and / are evaluated from left to right
- A good practice: use parentheses to make the operation order obvious

[Complete operation order in Python documentation](#)



What are the values of the following expressions?

- `print(3+4*2)`
- `print(2*3**2)`
- `print(6+4/2)`
- `print(1/2*2)`

What if you want to do multiplication first?

```

IDLE Shell 3.11.1
>>> print(2*3**2)
18
>>> print((2*3)**2)
36
Ln: 57 Col: 0

```

A good practice: use parentheses to make the operation order obvious



Assignment Operator (=)

- Assigns a value to a variable
- Generic form
 - Variable = expression
- Examples
 - $a=12$
 - $a=23+3$
 - $a=a*4$
 - $a=a*a+a$

Expression: Combination of literals, variables and operators



Assignment Operators

- Update the variable to a new value using the **existing** old value.

Operator	Example	Equivalent
<code>+=</code>	<code>a += 2</code>	<code>a = a + 2</code>
<code>-=</code>	<code>a -= 5</code>	<code>a = a - 5</code>
<code>*=</code>	<code>b *= 2</code>	<code>b = b * 2</code>
<code>/=</code>	<code>b /= 4</code>	<code>b = b / 4</code>
<code>%=</code>	<code>b %= 2</code>	<code>b = b % 2</code>
<code>//=</code>	<code>c //= 3</code>	<code>c = c // 3</code>
<code>**=</code>	<code>c **= 2</code>	<code>c = c ** 2</code>

```
IDLE Shell 3.10.0
>>> a = a + 1
Traceback (most recent call last):
  File "<pyshell#45>", line 1, in <module>
    a = a + 1
NameError: name 'a' is not defined
>>> a = 0
>>> a = a + 1
>>> print(a)
1
>>> a += 1
>>> print(a)
2
>>> |
Ln: 86 Col: 0
```



Mixed Types

- When an expression involves different types, what type will be used
 - During the calculation
 - As the final result
- Consider

```
a=1.5
```

```
b=3
```

```
c=a*b
```

```
print(c)
```

4?

4.5?

3?



What Python does?

- Converts operands up to the type of the most complex operand
- Performs the math on same-type operands
- Result will be in most complex operand type
- Complex > Float > Integer
- Example:
 - Convert `b` to float
 - Perform multiplication on `a` and `b`, both are float
 - Result will be in float
 - So, the output is 4.5

Operand is the value on which an operator operates.

```
a=1.5  
b=3  
c=a*b  
print(c)
```



String

- A sequence of characters
- In Python, a string can be enclosed by single quotation marks ('...'), double quotation marks ("..."), or triple quotation marks ("..." or "...")
- `s='ENGG1330'`
- Code can access individual character
- `ch=s[2]`

0	1	2	3	4	5	6	7
E	N	G	G	1	3	3	0

```
>>> s = "ENGG1330"  
>>> ch = s[2]  
>>> print(ch)  
G  
>>> |
```



String - len

- Built-in function `len` that returns the number of characters in a string

```
s = "Programming"  
a = len(s)  
print(a)
```

```
>>> s="Programming"  
>>> a=len(s)  
>>> print(a)  
11
```



String Operation (+)

- Concatenate a string to another string, i.e., join strings end-to-end

```
s1 = "ENGG1330"  
s2 = " Programming I"  
s = s1 + s2
```

```
>>> s1="ENGG1330"  
>>> s2=" Programming I"  
>>> s = s1 + s2  
>>> print(s)  
ENGG1330 Programming I  
>>>
```



String Operation (*)

- Repeat a string a number of times

```
s = "ENGG1330"  
s3 = s*3
```

```

IDLE Shell 3.10.0
>>> s1='ENGG1330'
>>> s3=s1*3
>>> print(s3)
ENGG1330ENGG1330ENGG1330
>>> |
Ln: 111 Col: 0

```



Casting

```
bmi_wrong.py - /Users/csvlee/Documents/c...  
weight=input('weight=')  
height=input('height=')  
bmi=weight/height/height  
print(bmi)  
|
```

```
IDLE Shell 3.11.1  
= RESTART: /Users/csvlee/Documents/courses/1330/sample_programs/bmi_wrong.py =  
weight=70  
height=1.65  
Traceback (most recent call last):  
  File "/Users/csvlee/Documents/courses/1330/sample_programs/bmi_wrong.py", line 3, in <module>  
    bmi=weight/height/height  
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Ln: 68 Col: 0

- By default, the input of the built-in function `input()` is a string type.
- We have to convert them to float type before doing the calculation (type casting).



Casting

- Python provides functions to specify/convert the type of a variable

Function	Description	Example
int()	Constructs an integer	int(2.8) int("4")
float()	Constructs a float number	float("3.2") float(2) float(4.5)
str ()	Constructs a string	str(2.3) str(3)



Casting

● ● ● bmi_right.py - /Users/csvlee/Documents/course...

```
weight=input('weight=')
print(type(weight))
weight=float(weight)
print(type(weight))
height=float(input('height='))
bmi=weight/height/height
print(bmi)
```

IDLE Shell 3.11.1

```
===== RESTART: /Users/csvlee/Documents/courses/1330/sample_programs/bmi_right.py =====
weight=70
<class 'str'>
<class 'float'>|
height=1.65
25.71166207529844
```

Ln: 77 Col: 15



How to control program printout?

- We control the output by formatting the output string.
- There are two ways to format a string:
 1. formatted string literals
 2. `str.format()`



Formatted String Literals

- Place `f` or `F` before the open quotation mark of a string
- Put Python expression inside a pair of braces `{ . . . }`
- The braces will be replaced by the values of Python expressions in the string

```
nofPeople=4  
time="7:00"  
print(f"We would like to reserve a \  
table for {nofPeople} at {time}")
```

If your code is too long,
this is the way to break
it into two lines



str.format()

- Enclose **formatting directives** with a pair of braces { . . . } where a variable will be substituted
- Provide variable value / expression with function `format`

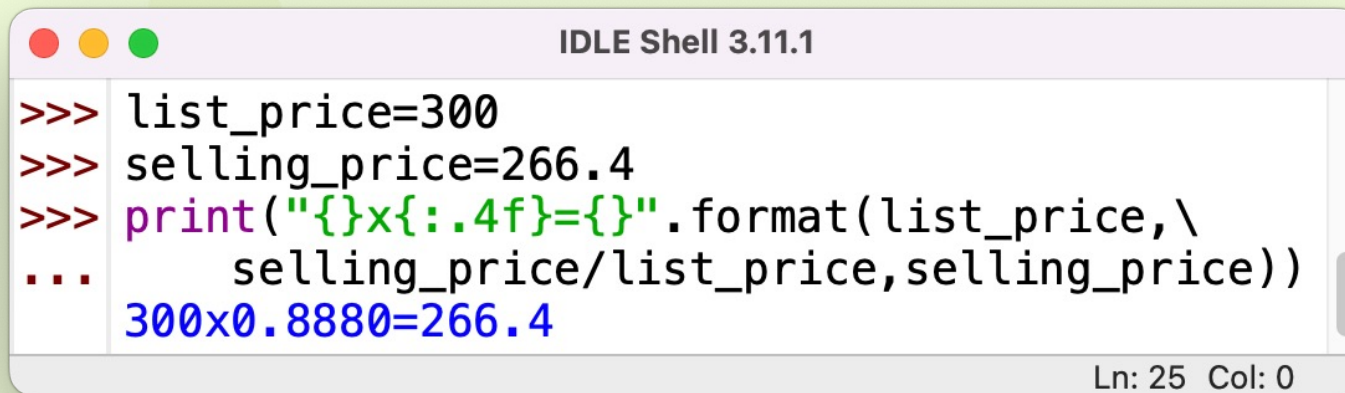
```
nofPeople=4  
time="7:00"  
print("We would like to reserve a \  
table for {} at {}".format(nofPeople, time))
```

These braces will be replaced
with the variables in `format`



More example of `str.format()`

```
list_price=300  
selling_price=266.4  
print("{}x{: .4f}={}".format(list_price, \  
selling_price/list_price,selling_price))
```



The screenshot shows an IDLE Shell window titled "IDLE Shell 3.11.1". The code from the previous block is entered and executed. The output is displayed on the last line: `300x0.8880=266.4`. The status bar at the bottom right indicates "Ln: 25 Col: 0".

```
>>> list_price=300  
>>> selling_price=266.4  
>>> print("{}x{: .4f}={}".format(list_price, \  
...     selling_price/list_price,selling_price))  
300x0.8880=266.4
```

Ln: 25 Col: 0



Format Syntax: Spacing

`{ : fill_char [>^<] width }`

```
ENG1330
      ENG1330
    ENG1330 |
ENG1330*****
+++++++ENG1330
-----ENG1330-----
```

Example	Description
<code>'{:<20}'.format('ENG1330')</code>	left aligned with 20 characters width
<code>'{:>20}'.format('ENG1330')</code>	right aligned with 20 characters width
<code>'{: ^20}'.format('ENG1330')</code>	centered with 20 characters width
<code>'{: *<20}'.format('ENG1330')</code>	left aligned with 20 characters width, fill space with '*'
<code>'{: +>20}'.format('ENG1330')</code>	right aligned with 20 characters width, fill space with '+'
<code>'{: - ^20}'.format('ENG1330')</code>	centered with 20 characters width, fill space with '-'



Format Syntax: Decimal places

{ : *width* . *decimal_places* f }

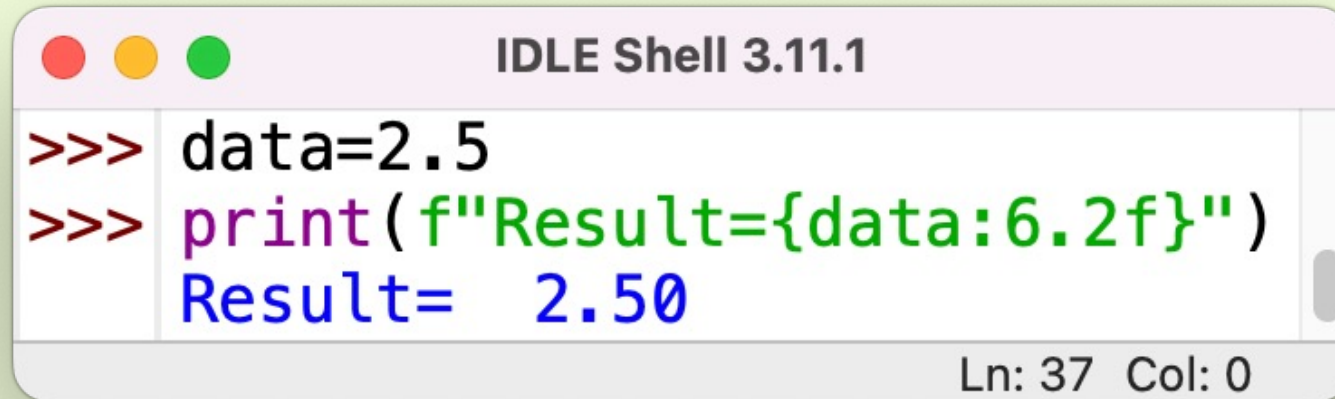
```
result=2.10
result= 2.10
result=  2.10
result= 2.123
result=**3.142
```

Example	Description
'result={:0.2f}'.format(2.1)	2 decimal places with data width
'result={:6.2f}'.format(2.1)	2 decimal places with 6 characters width
'result={:7.2f}'.format(2.1)	2 decimal places with 7 characters width
'result={:^9.3f}'.format(2.12345)	centered with 9 characters width
'result={:*>7.3f}'.format(3.14159)	right aligned with 7 characters width, fill space with '*'



Format Syntax in Formatted String Literals

```
data=2.5  
print(f"Result={data:6.2f}")
```



```
IDLE Shell 3.11.1  
>>> data=2.5  
>>> print(f"Result={data:6.2f}")  
Result= 2.50  
Ln: 37 Col: 0
```



More on String Formatting

- <https://docs.python.org/3/tutorial/inputoutput.html>



Summary

- Program uses variable to store value temporarily
- Variable should belong to one of the data types defined in the language
- Value assigned to a variable can be in form of literal or expression
- Python is a weakly type language, programmer must pay extra care on variable assignment and expression construction

