BASIC IO

Formatted Output & Console Input

Optional *print* Argument *sep*

Ex1_printArguments.py

- Consider statement
  `print(value0, value1, …, valueN)`
- *Print* function uses string consisting of **one space** character as **separator**
- Optionally change the separator to any string we like with the ***sep*** argument

Statement	Outcome
`print("Hello", "World!", sep="**")`	`Hello**World!`
`print("Hello", "World!", sep="")`	`HelloWorld!`
`print("1", "two", 3, sep="   ")`	`1   two   3`

Optional *print* Argument *end*

- Print statement ends by executing a **newline** operation.
- Optionally change the ending operation with the ***end*** argument

```
print("Hello", end=" ")
print("World!")
```

[Run]

```
Hello World!
```

```
print("Hello", end="")
print("World!")
```

[Run]

```
HelloWorld!
```

The *expandtabs()* Method

- *expandtab*() – controls the number of position between horizontal tab stops (default is 8)

```
## Demonstrate use of escape sequences.
print("01234567890123456")
print("a\tb\tc")
print("a\tb\tc".expandtabs(5))
print("Nudge, \tnudge, \nwink, \twink.".expandtabs(11))
```

[Run]

```
01234567890123456
a       b       c
a    b    c
Nudge,     nudge,
wink,      wink.
```

Justifying Output in a Field

Ex2_justification.py

- *ljust*(), *rjust*() and *center*() methods control the justification of data in a field of a specified width

```
## Demonstrate justification of output.
print("01234567890123456789012345678")
print("Rank".ljust(5), "Player".ljust(20), "HR".rjust(3), sep="")
print('1'.center(5), "Barry Bonds".ljust(20), "762".rjust(3), sep="")
print('2'.center(5), "Hank Aaron".ljust(20), "755".rjust(3), sep="")
print('3'.center(5), "Babe Ruth".ljust(20), "714".rjust(3), sep="")
```

[Run]

```
01234567890123456789012345678
Rank Player                HR
  1  Barry Bonds           762
  2  Hank Aaron            755
  3  Babe Ruth             714
```

The *format*() Method

Ex3a_format.py & Ex3b_format.py

- The *format*() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are
- The placeholder: curly brackets: { }
- zero-based numbering system - Numbering begins with zero instead of one
- More formatting types https://www.w3schools.com/python/ref_string_format.asp

```
fruit1 = “apples”

fruit2 = “pears”

print( “I want to buy {} and {}.”
.format(fruit1, fruit2) )

[Run]

I want to buy apples and pears.
```

Justify Output with *format* Method (1 of 4)

- Given: str1 is a string and w is a field width

```
print("{0:<ws}".format(str1))
print("{0:^ws}".format(str1))
print("{0:>ws}".format(str1))
```

```
print(str1.ljust(w))
print(str1.center(w))
print(str1.rjust(w))
```

Justify Output with *format* Method (2 of 4)

- Given: num is a number and w is a field width

```
print("{0:<wn}".format(num))
print("{0:^wn}".format(num))
print("{0:>wn}".format(num))
```

```
print(str(num).ljust(w))
print(str(num).center(w))
print(str(num).rjust(w))
```

Justify Output with *format* Method (3 of 4)

```
## Demonstrate justification of output.
print("01234567890123456789012345678")
print("{0:^5s}{1:<20s}{2:>3s}".format("Rank", "Player", "HR"))
print("{0:^5n}{1:<20s}{2:>3n}".format(1, "Barry Bonds", 762))
print("{0:^5n}{1:<20s}{2:>3n}".format(2, "Hank Aaron", 755))
print("{0:^5n}{1:<20s}{2:>3n}".format(3, "Babe Ruth", 714))
```

[Run]

```
01234567890123456789012345678
Rank  Player                HR
  1   Barry Bonds          762
  2   Hank Aaron           755
  3   Babe Ruth            714
```

Justify Output with *format* Method (4 of 4)

Ex3b_Format.py

- Number formatting

Statement	Outcome	Comment
print("{0:10d}".format(12345678))	12345678	number is an integer
print("{0:10,d}".format(12345678))	12,345,678	thousands separators added
print("{0:10.2f}".format(1234.5678))	1234.57	set precision to two decimal places
print("{0:10,.2f}".format(1234.5678))	1,234.57	set precision and separators added
print("{0:10,.3f}".format(1234.5678))	1,234.568	set precision and separators added
print("{0:10.2%}".format(12.345678))	1234.57%	set precision and % added
print("{0:10,.3%}".format(12.345678))	1,234.568%	set precision, separators and % added

F-strings (1 of 3)

- PEP (Python Enhancement Proposal) 498 introduced a new string **formatting** mechanism, literal string interpolation
- F-string, the leading ***f* character** preceding the string literal
- Making string interpolation simpler
- Providing a concise and convenient way to embed python expressions inside string literals for formatting

F-strings (2 of 3)

Ex4a_fstrings.py

- To create an f-string, prefix the string with the letter “f”
- f-strings can be used upon variables
- f-strings can span multiple lines if parenthesis are used
- https://www.geeksforgeeks.org/formatted-string-literals-f-strings-python/

```
fruit1 = “apples”

fruit2 = “pears”

qty = 7


txt = (f“I want to buy {qty} ”
f“{fruit1} and {fruit2}.”)

print(txt)

[Run]

I want to buy 7 apples and pears.
```

F-strings (3 of 3)

Ex4b_fstrings.py

- Align information into columns of specific width
- Justify columns left, right and/or center aligned

```
fruit1 = “apples”
brand1 = “gala”
qty = 100
print(f"{fruit1:<10}{brand1:>10}{qty:^10}")
[Run]
apples          gala   100
```

PE3_1

1. A – D, determine the output displayed by the lines of code. Save your code as *PE3_1.py*

A	`print("012345678901234567890")` `print('A'.rjust(5), 'B'.center(5), 'C'.ljust(5), sep = "")`
Output	

B	`print("01234567890123456")`
	`print("{0:^7}{1:4}{2:>6s}".format("one", "two", "three"))`
	`print(f"{'one':^7}{'two':4}{'three':>6s}")`

Output	

C	`n=1234`
	`print("01234567890123456789")`
	`print(f"{n:10}{n:^10}")`
	`print("01234567890123456789")`
	`print(f"{n:<10}{n:>10}")`
	`print("01234567890123456789")`
	`print(f"{n:10.3f}{n:10,.2f}{n}")`
	`print("012345678901234567890123456789")`
	`print(f"{n:10.2%}{n:12,.2%}")`

Output	

D	`q1= '''"If {0:s} dream it, {0:s} do it. - Walt Disney"'''`
	`print(q1.format('you can'))`
	
	`a = "ONE"`
	`b = "DAY"`
	`q2 = f"\"{a} {b} or {b} {a}. You decide. Paulo Coelho\""`
	`print(q2)`

Output	

The *input()* Function (1 of 2)

- Prompts the user to enter data

```
town = input("Enter the name of your city: ")
```

  - *User types response, presses ENTER key*
  - *Entry assigned to variable on left*

The *input()* Function (2 of 2)

Ex4_input.py

- The *input* function always returns a string
- A combination of an *input* function and an *int*, *float*, or *eval* function allows numbers to be input into a program

```
>>> letGrade = input("Enter your grade(A-F): ")
>>> numGrade = int(input("Enter your grade(0 - 100): "))
>>> average = float(input("Enter your average(0.0 - 100.0):"))
>>> gpa = eval(input("Enter your GPA(0 - 4): "))
```

PE3_2

2. Calculates the amount of a server's tip. Save the code as *PE3_2.py*.
a) Prompt and request input the amount of the bill (in float) and the percentage of tip (in integer).
b) Calculate, set the result to two decimal places and print the result.
Input text can be any content. Just make sure to precisely match the output format below.

```
Example Output 1:
Enter the amount of the bill: 36.99
Enter the percentage of tip: 18
Tip: $6.66
```

```
Example Output 2:
Enter the amount of the bill: 100
Enter the percentage of tip: 20
Tip: $20.00
```

PE3_3 & PE3_4 & PE3_5

- Write your codes and run

Summary (1 of 3)

- **`print(var1, …, varN, sep=str1, end=str2)`** displays the N values separated by str1 and ending with str2. The arguments ***sep*** and ***end*** are optional and have default values “ ” and “\n”.
- When the right the side of the colon in a pair of curly braces is just the letter s, the colon and the letter s can be omitted. For instance, {0:s} can abbreviated to {0}.
- A place holder such as {0} applies not only to strings, but also to numbers and expressions.

Summary (2 of 3)

- When the *format* method is used to format a string, **left-justify** is the default justification. Therefore, when a < (less than), ^ (caret), or > (greater than) symbol is not present, the string will be displayed left-justified in its field.
- When the *format* method is used to format a number, **right-justify** is the default justification.
- The *format* method replaces numbered placeholders of the form {n:format specifier} in a string with comma separated arguments of the method.

```
str1 = "The population of {0:s} is {1:.2%} of the U.S. population."
print(str1.format("Texas", 26448000 / 309000000))
```

Summary (3 of 3)

- The *Input* function displays a prompt and then assigns data entered by the user to a variable.

```
>>> gpa = eval(input("Enter your GPA(0 - 4): "))
```

- The *eval*() function evaluates the specified expression, if the expression is a legal Python statement, it will be executed.
- The *int* and *float* functions execute faster than *eval* function and are preferred by many Python programmers when they can be used safely.
- The *int* and *float* are constructor functions to perform type casting numbers.

```
>>> int(4.8)          >>> float(4)
4                     4.0
```

Variables - Terminologies

- ❑ 1 argument
- ❑ 2 separator
- ❑ 3 sep argument
- ❑ 4 newline
- ❑ 5 end argument
- ❑ 6 \n
- ❑ 7 \t
- ❑ 8 .expandtab(n)
- ❑ 9 optional
- ❑ 10 default
- ❑ 11 place holder
- ❑ 12 str.ljust(n)
- ❑ 13 str.rjust(n)
- ❑ 14 str.center(n)
- ❑ 15 ^
- ❑ 16 <
- ❑ 17 >
- ❑ 18 ,
- ❑ 19 .nf
- ❑ 20 .n%
- ❑ 21 prompt
- ❑ 22 input()
- ❑ 23 float()
- ❑ 24 int()
- ❑ 25 eval()
- ❑ 26 constructor functions
- ❑ 27 type casting
- ❑ 28 exception handling
- ❑ 29 try statement
- ❑ 30 exception clauses
- ❑ 31 indentation/tabs

Quiz 3

- Quiz 3A has 10 questions in 15 minutes, 10 pts
  - *10 multiple choice/true or false questions, 1 pt. for each question*
  - *Quiz 3A has two attempt, the higher grade will be selected*
  - *Submit Quiz 3A (at least 1-minute) **before** the due time to Blackboard*

- Quiz 3B has 2 code questions, 15 pts
  - *Write the Python code based on the given question*
  - *Each question will be given during the first 10-minute of each session of week 3*
  - *Quiz 3B-1 on session A, and Quiz 3B-2 on session B*
  - *Quiz 3B has one attempt*

DB 3

- Instruction:
  1) Choose any **one** of the questions from **PE3_1**, any **one** of the questions from **PE3_6**, **and** any **one** of the questions from **PE3_7**. Please **avoid** selecting the exact same questions. Make sure to indicate the **question #** you're working on in the thread title as soon as you open your thread. Then you can **explain and edit your questions** (1.2 pts).
  2) After posting your explanation, check each other's answers, and ask questions or make comments (0.3 pts).
  3) Submit your posts before the due date. Let's learn from each other.