

파이썬 개발환경과 가상환경

컴퓨터정보공학과
강 환수 교수

다양한 개발환경과 가상환경 소개

- 파이썬의 다양한 개발환경
 - 표준 개발환경
 - IDLE 셸
 - 통합개발환경
 - Pycharm, Spyder 등
 - 웹 개발환경, 노트북
 - ipython 기반, 파일 확장자 ipynb
 - 주피터 노트북, 코랩 노트북, 캐글 노트북
 - 전문 에디터 개발환경
 - 비주얼 스튜디오 코드, 서브라임 텍스트, 아톰 등
- 가상 환경
 - 독립적인 파이썬 개발 환경
 - 하나의 컴퓨터에 여러 개 생성 가능

수업 개요

- 2020-5/7(목)에서 6/11까지 6주 동안 매주 목, 총 36시간
 - 2시간 수업, 1시간 질의 및 자기주도학습
- 교재
 - 일부 활용:
 - 파이썬에 참 좋은 파이참
 - 코딩 참조:
 - 파이썬으로 시작하는 컴퓨터과학입문
 - 파이썬프로그래밍개론(An Introduction to Programming using Python)
 - Realpython.com
- 일정(변경될 수 있음)

		오전	오후
2020-05-07(목)	1일차	파이썬 기본 개발 환경	파이썬 문법 코딩하기(기본과 함수, 리스트)
2020-05-14(목)	2일차	아나콘다 설치, 주피터 노트북 사용	파이썬 문법 코딩하기(딕셔너리와 튜플, 집합)
2020-05-21(목)	3일차	모듈 기본, 파이참 설치	파이썬 문법 코딩하기, import this 코딩
2020-05-28(목)	4일차	가상환경 개요와 생성	직접 가상환경 생성과 파이참에서 생성
2020-06-04(목)	5일차	주피터 노트북과 파이참 심화	구글 코랩
2020-06-11(목)	6일차	비주얼 스튜디오 코드	서브라임텍스트와 아톰

파이썬의 표준 개발환경

파이썬 설치 후 메뉴

- **IDLE**
 - IDE 모습의 쉘
- **Python shell**
 - 도스 창 모습의 쉘
- **파이썬 매뉴얼**
 - 매뉴얼
- **Module docs**
 - 설치 모듈 문서
- **모듈**
 - 라이브러리 파일
 - 내부에는 함수, 클래스 등의 파이썬 소스



IDLE 쉘

• 주요 시스템 환경 조사

- sys.prefix: 현 파이썬 설치 루트 폴더
- sys.version, sys.platform: 버전과 플랫폼
- sys.path: 현 파이썬 실행의 경로 목록
 - **sys의 path는 실행파일이나 패키지를 찾는 순서**
 - 이 경로에 없으면 오류 발생

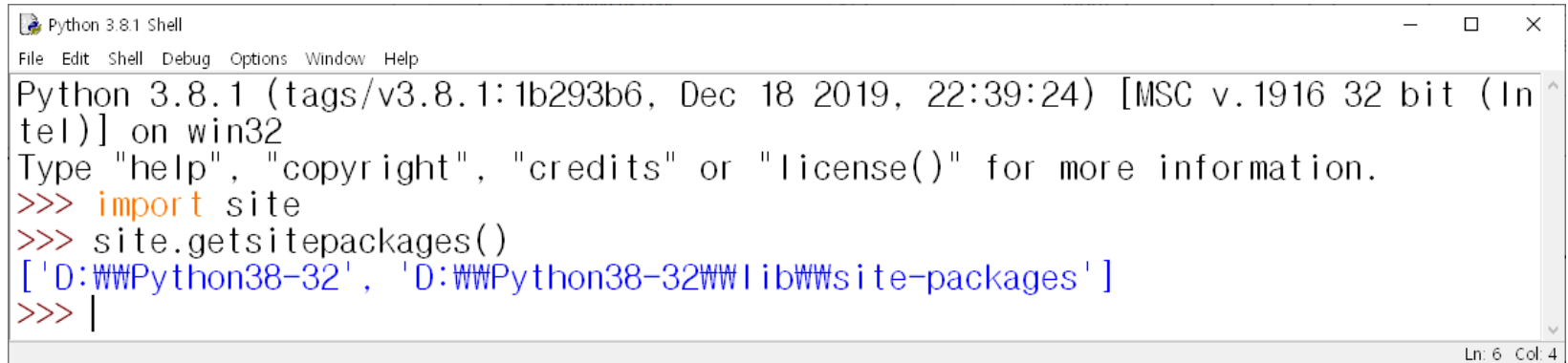
```

Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import sys
>>> sys.prefix
'D:\\Python38-32'
>>> sys.version
'3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)]'
>>> sys.platform
'win32'
>>> sys.path
['', 'D:\\Python38-32\\Lib\\idlelib', 'D:\\Python38-32\\python38.zip', 'D:\\Python38-32\\DLLs', 'D:\\Python38-32\\Lib', 'D:\\Python38-32', 'C:\\Users\\217\\AppData\\Roaming\\Python\\Python38\\site-packages', 'D:\\Python38-32\\Lib\\site-packages', 'D:\\Python38-32\\Lib\\site-packages\\win32', 'D:\\Python38-32\\Lib\\site-packages\\win32\\Lib', 'D:\\Python38-32\\Lib\\site-packages\\Pythonwin']
>>>
  
```

패키지 설치 폴더

- `site.getsitepackages()`

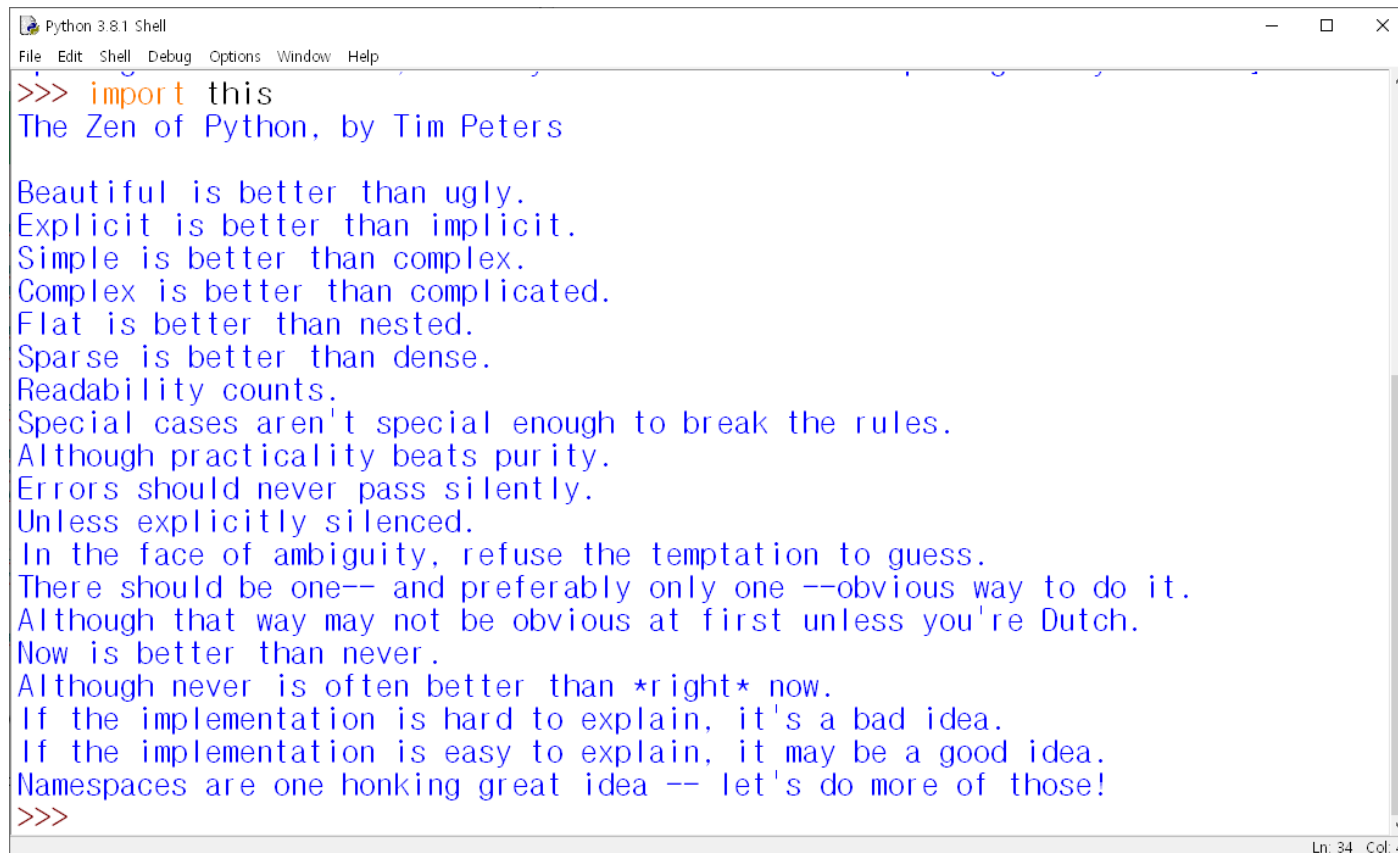
```
>>> import site
>>> site.getsitepackages()
['D:\WWPython38-32', 'D:\WWPython38-32\lib\site-packages']
>>>
```



```
Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import site
>>> site.getsitepackages()
['D:\WWPython38-32', 'D:\WWPython38-32\lib\site-packages']
>>> |
```

파이썬의 철학

- import this

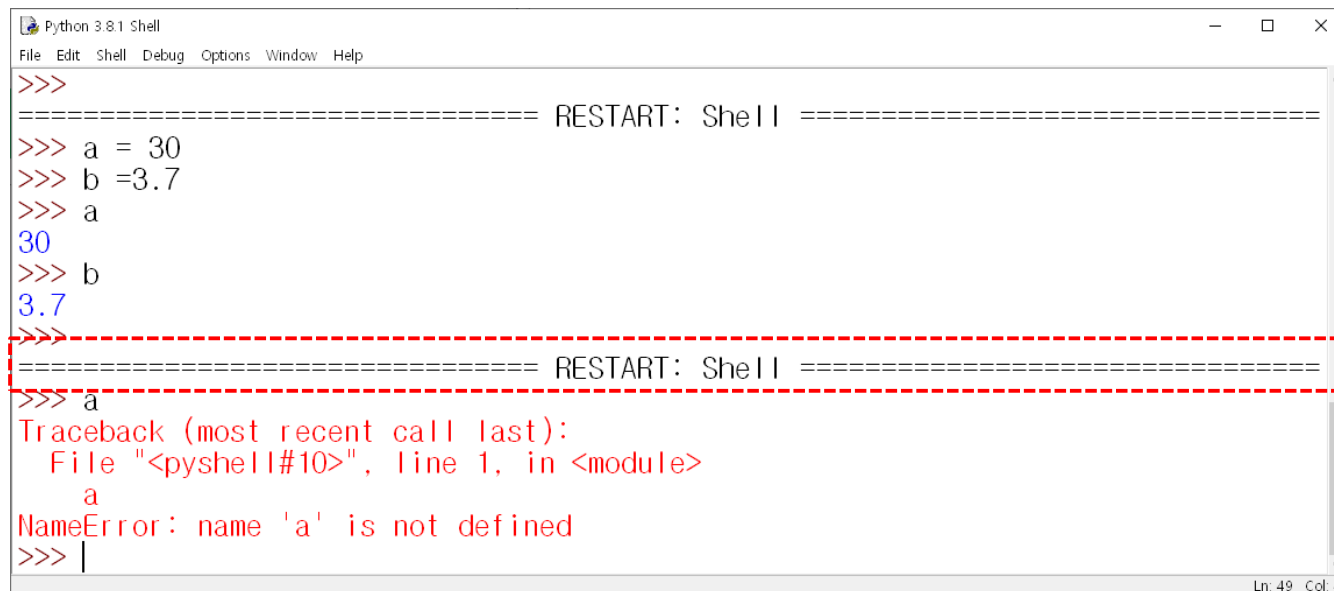


```
Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```


파이썬 재실행

- 셸은 위에서 실행된 내용을 저장
- 재실행
 - 메뉴 Shell | Restart shell, ctrl + F6
 - 모든 정보가 사라지고 다시 시작



The screenshot shows a Python 3.8.1 Shell window with a menu bar (File, Edit, Shell, Debug, Options, Window, Help). The shell contains the following text:

```
>>>
===== RESTART: Shell =====
>>> a = 30
>>> b = 3.7
>>> a
30
>>> b
3.7
>>>
===== RESTART: Shell =====
>>> a
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    a
NameError: name 'a' is not defined
>>> |
```

A red dashed rectangle highlights the second restart section, from the line "===== RESTART: Shell =====" down to the line ">>> |".

Ln: 49 Col: 4

주요 기능

- 코드 히스토리
 - 이전 코드
 - alt + P
 - 이후 코드
 - alt + N
- 코드 이후 보이기
 - 코딩 시 ctrl + space
- 도움말
 - help(print)

```
>>> help(print)
Help on built-in function print in module builtins:

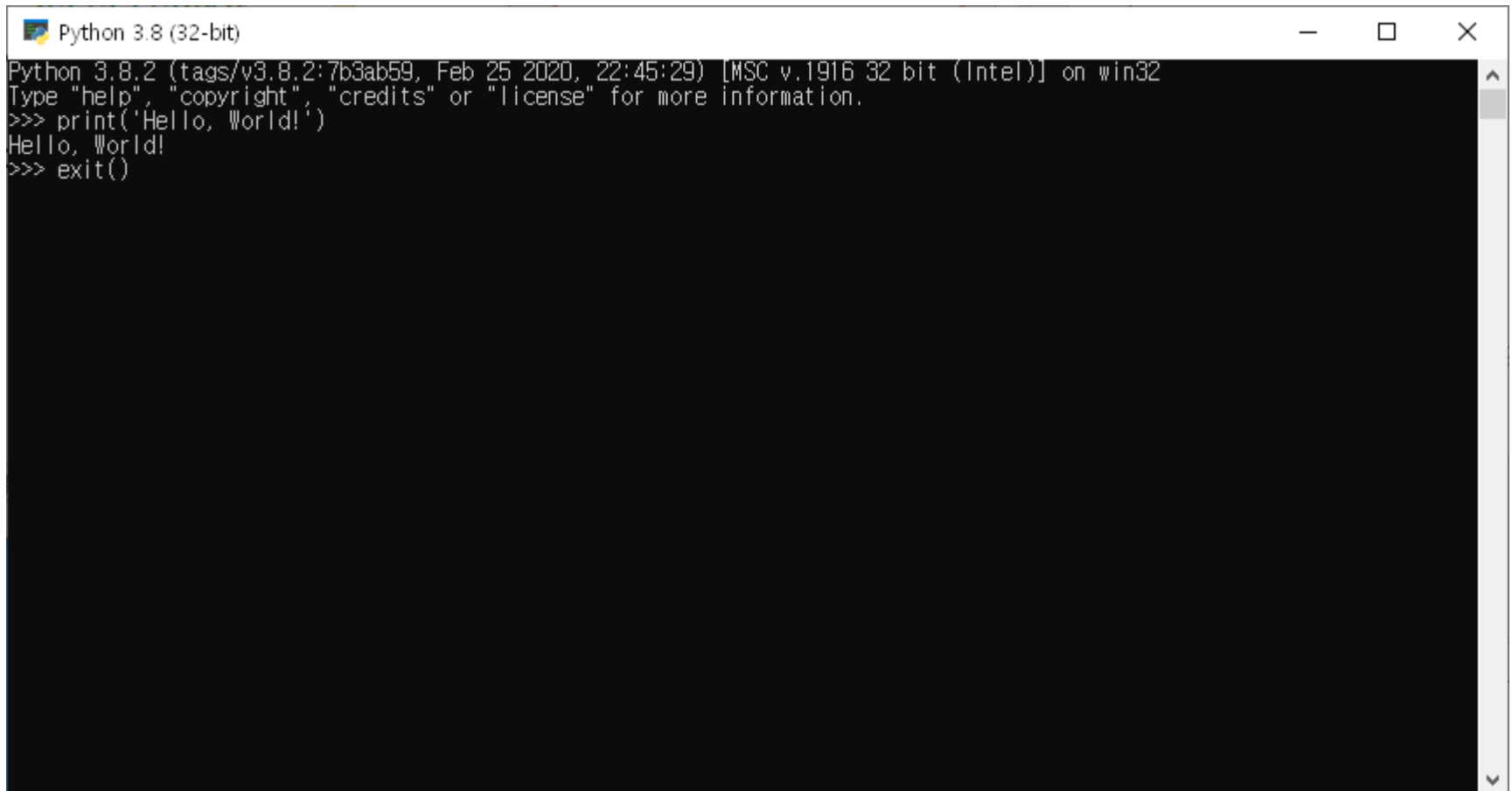
print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file: a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

도스창에서 직접 개발

파이썬 셸

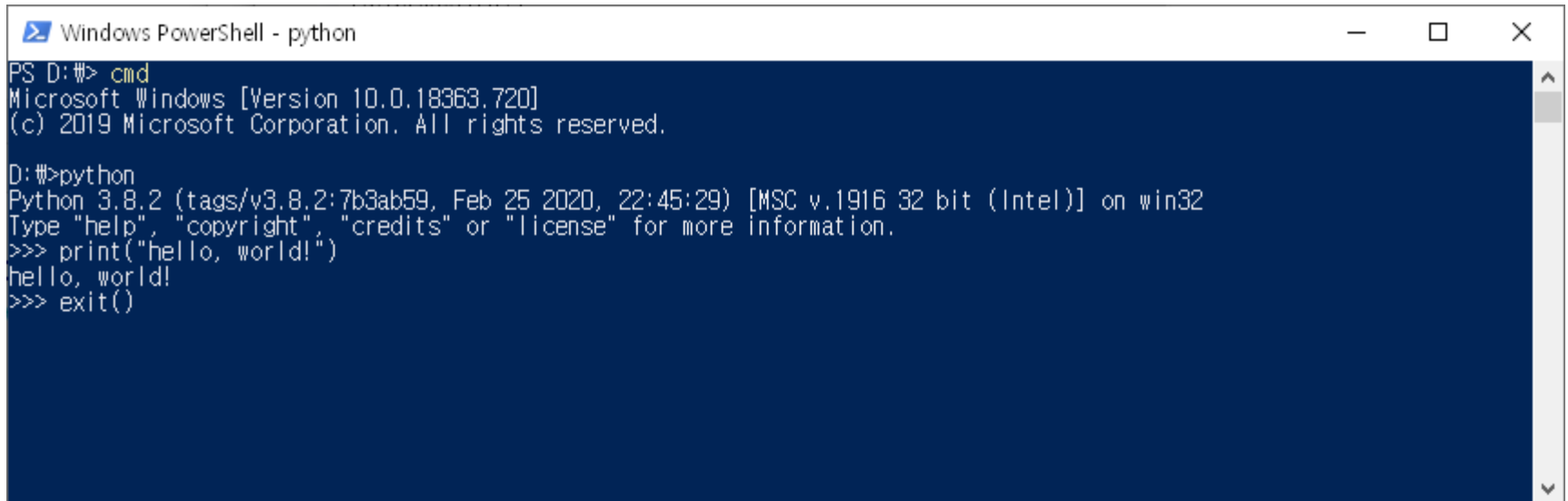
- 셸 나가기
 - ctrl+D, exit()



```
Python 3.8 (32-bit)
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello, World!')
Hello, World!
>>> exit()
```

명령어 python

- 도스창에서 파이썬 셸 실행



```
Windows PowerShell - python
PS D:\> cmd
Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\>python
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello, world!")
hello, world!
>>> exit()
```

파일에 저장, hello.py

- 직접 파이썬 소스인 파일 실행
 - python hello.py



The screenshot shows a Windows PowerShell window with the following commands and output:

```
>>> exit()  
D:\>notepad hello.py  
D:\>python hello.py  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
D:\>
```

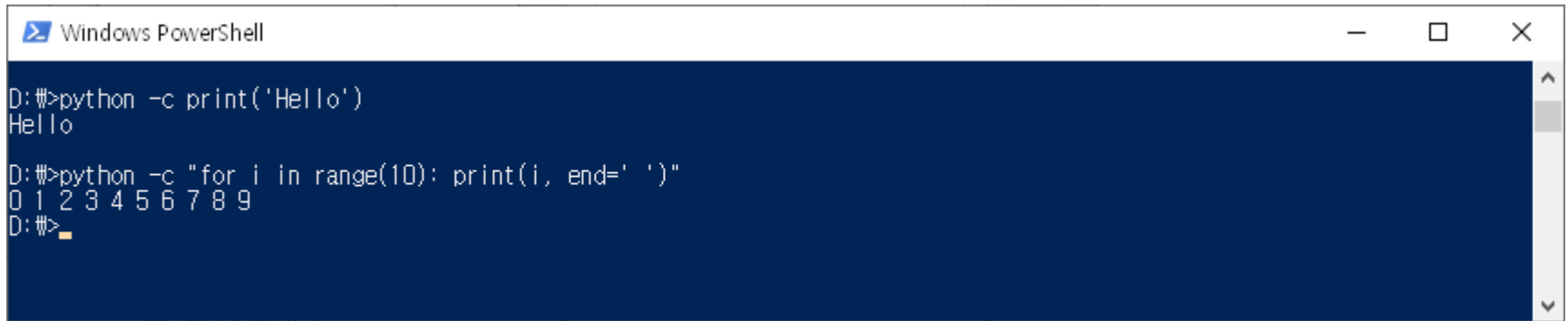
Overlaid on the PowerShell window is a Notepad window titled "hello.py - Windows 메모장". The Notepad window contains the following Python code:

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

The Notepad window also shows a status bar at the bottom indicating "Ln 2, Col 13", "100%", "Windows (CRLF)", and "UTF-8".

한 줄 실행

- 옵션 -c



```
Windows PowerShell
D:\>python -c print('Hello')
Hello
D:\>python -c "for i in range(10): print(i, end=' ')"
0 1 2 3 4 5 6 7 8 9
D:\>
```

- `python -c "i = 10; print('짝수') if i%2 == 0 else print('홀수')"`

파이썬 매뉴얼

Python 3.8.2 documentation

handles multiple arguments, floating point quantities, and strings. Strings are printed without quotes, and a space is inserted between items, so you can format things nicely, like this:

```
>>> i = 256*256
>>> print('The value of i is', i)
The value of i is 65536
```

The keyword argument `end` can be used to avoid the newline after the output, or end the output with a different string:

```
>>> a, b = 0, 1
>>> while a < 1000:
...     print(a, end=',')
...     a, b = b, a+b
...
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,
```

Footnotes

- [1] Since `**` has higher precedence than `-`, `-3**2` will be interpreted as `-(3**2)` and thus result in `-9`. To avoid this and get `9`, you can use `(-3)**2`.
- [2] Unlike other languages, special characters such as `\n` have the same meaning with both single (`'...'`) and double (`"..."`) quotes. The only difference between the two is that within single quotes you don't need to escape `"` (but you have to escape `'`) and vice versa.

Python » 3.8.2 Documentation » The Python Tutorial »

previous | next | modules | index

© Copyright 2001-2020, Python Software Foundation.
The Python Software Foundation is a non-profit corporation. [Please donate.](#)

Last updated on Feb 25, 2020. [Found a bug?](#)
Created using [Sphinx](#) 2.4.3.

IDLE 쉘에서 프로그래밍 연습

작업할 폴더 설정

- 다음 폴더를 생성
 - Python IDE & VE
 - **Lect0507** 폴더 하부
 - 위 폴더 하부에 소스 파일 생성

문자열과 함수

- 문자열
- 함수
 - 출력
 - 입력
- 제어문
 - for 반복
- 들여쓰기

```
print("This program illustrates a chaotic function")
x = float(input("Enter a number between 0 and 1: "))
#x = eval(input("Enter a number between 0 and 1: "))
for i in range(10):
    x = 3.9 * x * (1 - x)
    print(x)
```

주석

- 함수 정의와 호출

```
# avg2.py
#   A simple program to average two exam scores
#   Illustrates use of multiple input

def main():
    print("This program computes the average of two exam scores.")

    score1, score2 = eval(input("Enter two scores separated by a comma: "))
    average = (score1 + score2) / 2

    print("The average of the scores is:", average)

main()
```

```
print("This program computes the average of two exam scores.")

score1, score2 = eval(input("Enter two scores separated by a comma: "))
average = (score1 + score2) / 2

print("The average of the scores is:", average)
```

```
1  # convert.py
2  #      A program to convert Celsius temps to Fahrenheit
3  # by: Susan Computewell
4
5
6  def main():
7      celsius = eval(input("What is the Celsius temperature? "))
8      fahrenheit = 9 / 5 * celsius + 32
9      print("The temperature is", fahrenheit, "degrees Fahrenheit.")
10
11
12  main()
```

```
1  # futval.py
2  #   A program to compute the value of an investment
3  #   carried 10 years into the future
4
5
6  def main():
7      print("This program calculates the future value")
8      print("of a 10-year investment.")
9
10     principal = eval(input("Enter the initial principal: "))
11     apr = eval(input("Enter the annual interest rate: "))
12
13     for i in range(10):
14         principal = principal * (1 + apr)
15
16     print("The value in 10 years is:", principal)
17
18
19  main()
```

```
1  # change.py
2  #   A program to calculate the value of some change in dollars
3
4
5  def main():
6      print("Change Counter")
7      print()
8      print("Please enter the count of each coin type.")
9      quarters = eval(input("Quarters: "))
10     dimes = eval(input("Dimes: "))
11     nickels = eval(input("Nickels: "))
12     pennies = eval(input("Pennies: "))
13     total = quarters * 0.25 + dimes * 0.10 + nickels * 0.05 + pennies * 0.01
14     print()
15     print("The total value of your change is", total)
16
17
18     main()
```

```
1  # change2.py
2  #   A program to calculate the value of some change in dollars
3  #   This version avoids the eval function
4
5
6  def main():
7      print("Change Counter")
8      print()
9      print("Please enter the count of each coin type.")
10     quarters = int(input("Quarters: "))
11     dimes = int(input("Dimes: "))
12     nickels = int(input("Nickels: "))
13     pennies = int(input("Pennies: "))
14     total = 0.25 * quarters + 0.10 * dimes + 0.05 * nickels + 0.01 * pennies
15     print()
16     print("The total value of your change is", total)
17
18
19     main()
```



```
1  # factorial.py
2  #   Program to compute the factorial of a number
3  #   Illustrates for loop with an accumulator
4
5
6  def main():
7      n = int(input("Please enter a whole number: "))
8      fact = 1
9      for factor in range(n, 1, -1):
10         fact = fact * factor
11     print("The factorial of", n, "is", fact)
12
13
14  main()
```

```
1  # quadratic.py
2  #   A program that computes the real roots of a quadratic equation.
3  #   Illustrates use of the math library.
4  #   Note: this program crashes if the equation has no real roots.
5
6  import math # Makes the math library available.
7
8
9  def main():
10     print("This program finds the real solutions to a quadratic")
11     print()
12
13     a = float(input("Enter coefficient a: "))
14     b = float(input("Enter coefficient b: "))
15     c = float(input("Enter coefficient c: "))
16
17     discRoot = math.sqrt(b * b - 4 * a * c)
18     root1 = (-b + discRoot) / (2 * a)
19     root2 = (-b - discRoot) / (2 * a)
20
21     print()
22     print("The solutions are:", root1, root2)
23
24
25  main()
```