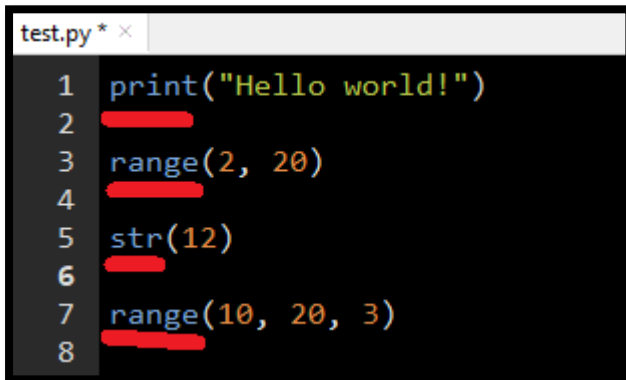


Functions

Program ရေးတဲ့အခါမှာ ရေးပြီးထားတဲ့ကုဒ်တွေကို ထပ်ခါထပ်ခါ ပြန်မရေးရဖို့အတွက် **function** ကို အသုံးပြုပါတယ်။

ပြီးခဲ့တဲ့သင်ခန်းစာတွေမှာ Python Language မှာပါဝင်တဲ့ မူလ **function** တွေ အတော်များများကို အသုံးပြုခဲ့ပြီးပါပြီ။



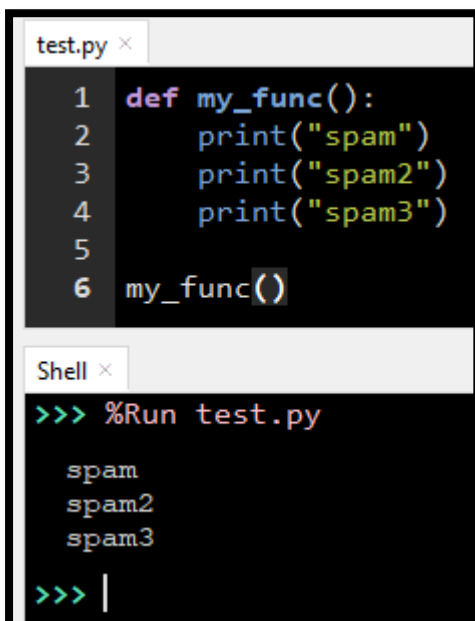
```

1 print("Hello world!")
2
3 range(2, 20)
4
5 str(12)
6
7 range(10, 20, 3)
8

```

အနိမ့်တားပြထားတာတွေက **function** တွေဖြစ်ပြီး () ထဲမှာဖော်ပြထားတာတွေက **function** တွေထဲမှာ ထည့်သွင်းအသုံးပြုတဲ့ တန်ဖိုး (function ရဲ့ **arguments**) တွေ ဖြစ်ပါတယ်။

Python Language မှာ ပါဝင်တဲ့ မူလ **function** တွေကို အသုံးပြုနိုင်သလို မိမိဘာသာလည်း **def** keyword ကို အသုံးပြုပြီး ကိုယ်ပိုင် **function** တည်ဆောက်နိုင်ပါတယ်။ အောက်ပါပုံကို ကြည့်ပါ။



```

1 def my_func():
2     print("spam")
3     print("spam2")
4     print("spam3")
5
6 my_func()

```

```

>>> %Run test.py
spam
spam2
spam3
>>> |

```

function ကို မိမိဘာသာဖန်တီးပြီးနောက် အသုံးပြုဖို့အတွက် **line 6** မှာလို မိမိရဲ့ **function** ကို ခေါ်ယူအသုံးပြုဖို့ လိုပါတယ်။ အခုနမူနာ **function** က ထည့်သွင်းအသုံးပြုရမဲ့ တန်ဖိုးမပါတဲ့အတွက် **function** ကို ခေါ်ယူအသုံးပြုတဲ့အခါမှာ **argument** မပါပါဘူး။ () ထဲမှာ ဘာမှမရေးထားတာကို တွေ့ရမှာပါ။

argument ထည့်သွင်းရမဲ့ **function** နမူနာပြပါမယ်။

```
test.py ×
1 def print_with_exclamation(word):
2     print(word + "!")
3
4 print_with_exclamation("spam")
5 print_with_exclamation("eggs")
6 print_with_exclamation("python")
7

Shell ×
>>> %Run test.py
spam!
eggs!
python!
>>> |
```

ဒီ **print_with_exclamation()** function က **argument** တန်ဖိုး တစ်ခု ထည့်သွင်းဖို့လိုအပ်တဲ့ function ဖြစ်ပါတယ်။ function အသုံးပြုတဲ့အခါမှာ () ထဲမှာ ထည့်သွင်းလိုက်တဲ့ **argument** တန်ဖိုးက **word** နေရာကို ရောက်သွားပြီး function ထဲမှာပါတဲ့ လုပ်ဆောင်ချက်ဖြစ်တဲ့ **word + "!"** ကို ဆက်လက်လုပ်ဆောင်ပါတယ်။ ဒီအတွက်ကြောင့် **spam!**, **eggs!**, **python!** တို့ကို မြင်ရခြင်း ဖြစ်ပါတယ်။

argument တစ်ခုထက်မက ပါဝင်တဲ့ function တွေကိုလည်း ဖန်တီးနိုင်ပါတယ်။

```

test.py x
1 def print_sum_twice(x, y):
2     print(x + y)
3     print(x + y)
4
5 print_sum_twice(5, 8)
6

Shell x
>>> %Run test.py
13
13
>>> |

```

function arguments တွေကို function ရဲ့လုပ်ဆောင်ချက်ထဲမှာ variable ပုံစံမျိုးနဲ့ အသုံးပြုနိုင်ပါတယ်။ ဒါပေမဲ့ function ထဲက variable ကို function ရဲ့ ပြင်ပကနေ ရယူအသုံးပြုလို့မရပါဘူး။ အောက်ပါနမူနာကို ကြည့်ပါ။

```

test.py x
1 def new_function (x_variable):
2     x_variable = x_variable + 2
3     print(x_variable)
4
5 new_function(7)
6 print(x_variable)
7

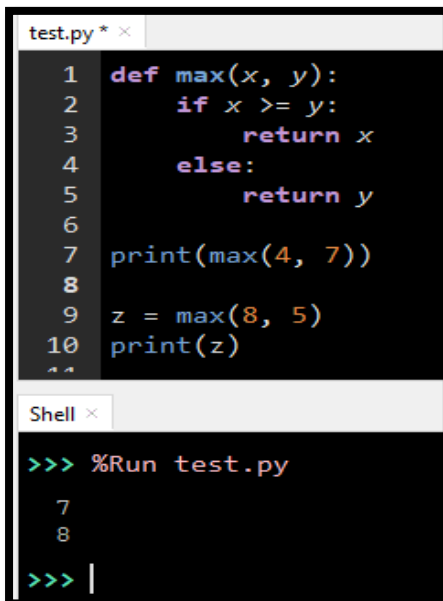
Shell x
>>> %Run test.py
9
Traceback (most recent call last):
  File "D:\Job\Python Myanmar Syllabus\test.py", line 6, in <module>
    print(x_variable)
NameError: name 'x_variable' is not defined
>>> |

```

Line 5 အတွက် ရလဒ်ထွက်ပေါ်လာပေမဲ့ Line 6 အတွက် NameError တက်နေတာကို တွေ့ရမှာပါ။

Returning from Functions

`int()` နဲ့ `str()` function တွေလိုမျိုး အချို့ function တွေက နောက်ပိုင်းမှာအသုံးပြုဖို့ တန်ဖိုးအသစ်တွေ ထုတ်ပေးကြပါတယ်။ `float` ဒေတာကို `int()` ထဲထည့်လိုက်ရင် `integer` ဒေတာအဖြစ် ပြန်ထွက်လာတာမျိုးကို ဆိုလိုပါတယ်။ မိမိဘာသာရေးတဲ့ function တွေမှာ အဲ့ဒီလို လုပ်ဆောင်လို့ရစေချင်တယ်၊ တန်ဖိုးတစ်ခုခု ပြန်ထုတ်ပေးစေချင်တယ်ဆိုရင် `return` keyword ကို အသုံးပြုပါတယ်။

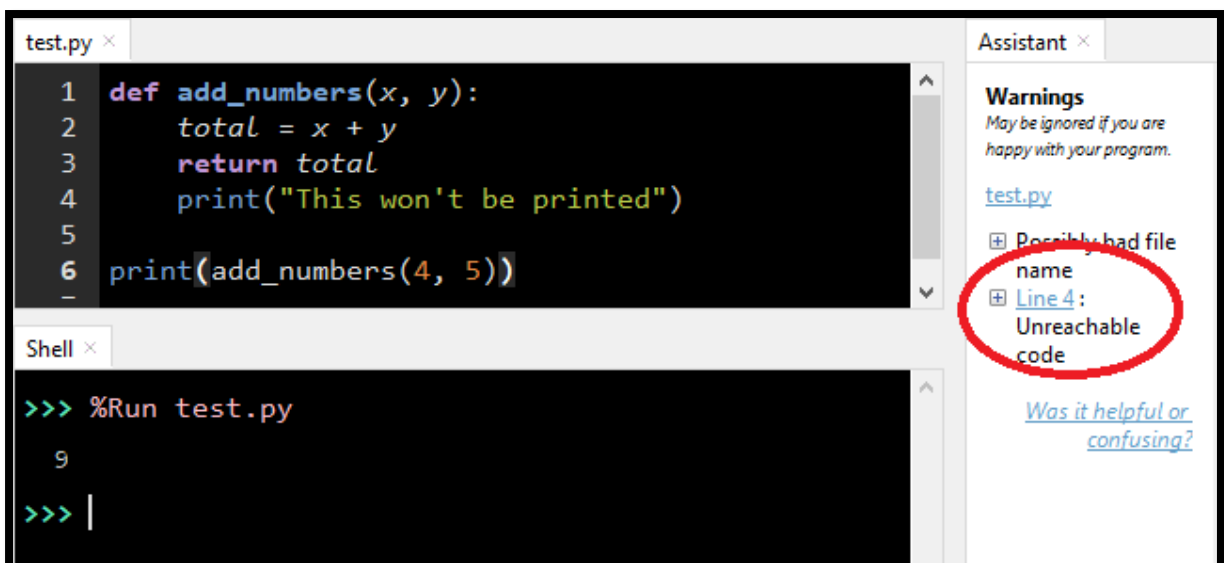


```
test.py * x
1 def max(x, y):
2     if x >= y:
3         return x
4     else:
5         return y
6
7 print(max(4, 7))
8
9 z = max(8, 5)
10 print(z)
11

Shell x
>>> %Run test.py
7
8
>>> |
```

(`return` ကို function ရဲ့ အပြင်ဘက်မှာ အသုံးပြုလို့မရပါဘူး)

function ထဲမှာပါတဲ့ လုပ်ဆောင်ချက်တွေကို လုပ်ဆောင်နေရင်း `return` ကို ရောက်လာလို့ တန်ဖိုးတစ်ခုခု ထုတ်ပေးပြီးတာနဲ့ တပြိုင်နက် `return` ရဲ့အောက်က စာကြောင်းတွေမှာပါတဲ့ လုပ်ဆောင်ချက်တွေကို လုံးဝလုပ်မပေးတော့ပါဘူး။ အောက်ပါနမူနာကို ကြည့်ပါ။



```
test.py x
1 def add_numbers(x, y):
2     total = x + y
3     return total
4     print("This won't be printed")
5
6 print(add_numbers(4, 5))

Shell x
>>> %Run test.py
9
>>> |
```

Assistant x

Warnings
May be ignored if you are happy with your program.

[test.py](#)

- ⊕ Possibly had file name
- ⊕ **Line 4:** Unreachable code

[Was it helpful or confusing?](#)

Comments

Comment ဆိုတာက မိမိပရိုဂရမ်ရဲ့ လုပ်ဆောင်ချက်ကို အကျဉ်းချုပ်ဖော်ပြဖို့ အမှတ်အသားအဖြစ်ပြုလုပ်ဖို့အတွက် အသုံးပြုပါတယ်။ # ကို အသုံးပြုပြီး ရေးသားပါတယ်။ # ရဲ့နောက်မှာ ရေးသားသမျှ စာသားတွေကို ပရိုဂရမ်က လုံးဝထည့်ပြီး အသုံးမပြုပါဘူး။ အောက်ပါနမူနာကို ကြည့်ပါ။

```
test.py x
1 x = 365
2 y = 7
3 # this is a comment
4
5 print(x % y) #find the remainder
6 # print(x//y)
7 # another comment

Shell x
>>> %Run test.py
1
>>> |
```

Docstrings

Docstrings (**Documentation Strings**) က comment ရဲ့ သဘောတရားနဲ့တူပြီး များသောအားဖြင့် function ရဲ့ လုပ်ဆောင်ချက်ကို ရှင်းပြဖို့အတွက် def ကိုအသုံးပြုပြီး function ကို ဖန်တီးလိုက်တဲ့ ပထမဆုံးစာကြောင်းရဲ့အောက်မှာ ရေးလေ့ရှိပါတယ်။ comment ထက်အားသာတဲ့အချက်က docstring ထဲမှာ စာကြောင်းတွေကို အကြောင်းရေအများကြီး ခွဲရေးလို့ရခြင်းဖြစ်ပါတယ်။

```
test.py x
1 def shout(word):
2     """
3     Print a word with an
4     exclamation mark following it.
5     """
6     print(word + "!")
7
8 shout("spam")

Shell x
>>> %Run test.py
spam!
>>> |
```

Functions as Objects

Function တွေကိုဖန်တီးတဲ့အခါမှာ ပုံမှန် variable ဖန်တီးတာနဲ့ မတူပေမဲ့ သူတို့ကို ပုံမှန် variable တွေလိုပဲ နာမည် (variable name) ပြောင်းလဲသတ်မှတ်ပြီး အသုံးပြုနိုင်ပါတယ်။

```

test.py x
1  def multiply(x, y):
2      return x * y
3
4  a = 4
5  b = 7
6  operation = multiply
7  print(operation(a, b))
8  print(multiply(a, b))

Shell x
>>> %Run test.py
28
28
>>> |
  
```

ပုံမှန် **multiply** function ကို **operation** ဆိုတဲ့အမည်နဲ့ variable အဖြစ် သတ်မှတ်ပေးလိုက်တာကို တွေ့ရမှာပါ။ ဒီအတွက်ကြောင့် **operation** ကို အသုံးပြုပြီး **multiply** function ရဲ့ လုပ်ဆောင်ချက်တွေကို ရယူအသုံးပြုနိုင်တာကို မြင်ရမှာပါ။ ဒီလို variable အမည်သစ်နဲ့ အသုံးပြုတဲ့အတွက် မူလ function ကို ထိခိုက်မှုမရှိပါဘူး။ line 8 ကို ကြည့်နိုင်ပါတယ်။

function တစ်ခုကို အခြား function တစ်ခုရဲ့ argument အဖြစ်လည်း အသုံးပြုနိုင်ပါတယ်။ အောက်ပါနမူနာပုံကို ကြည့်ပါ။

```
test.py ×
1 def add(x, y):
2     return x + y
3
4 def do_twice(function, x, y):
5     return function(function(x,y), function(x,y))
6
7 a = 5
8 b = 10
9
10 print(do_twice(add, a, b))

Shell ×
>>> %Run test.py
30
>>> |
```

ပုံမှန်ဖော်ပြထားတဲ့အတိုင်း **do_twice** function က **add** function ကို ရယူပြီး သူ့ရဲ့ လုပ်ဆောင်ချက်တွေအတွင်းမှာ ထည့်သွင်းအသုံးပြုတာကို မြင်ရမှာပါ။