

프로그래밍 자습서 > 프로그래밍 자습서 정보

프로그래밍 자습서 정보

PTC Mathcad 프로그래밍을 사용하면 여러 줄로 구성된 함수를 작성하고 반복 계산을 정의할 수 있습니다. 또한 복잡한 비교와 분기를 정의하고 값을 구할 수 있습니다. 프로그램을 정의할 때 PTC Mathcad 식을 조합하여 사용할 수 있으며 반복을 통해 간결성과 효율성을 높일 수 있습니다. 프로그래밍 연산자를 사용하면 변수 또는 함수에 대해 부분 변수 지정을 수행하고, 루프로 계산을 반복하고, 조건에 따라 분기를 계산하고, 중단점을 추가하고, 오류를 처리하고, 값을 반환할 수 있습니다.

풀이 자습서는 다음과 같은 순차적인 3개의 연습으로 구성되어 있습니다.

- 연습 1: 프로그램 시작 및 기본 연산자 사용
- 연습 2: 분기 작성(if-else 문)
- 연습 3: 루프 사용

연습은 순서대로 완료해야 합니다. [연습 1로 이동합니다.](#)

프로그래밍 자습서 > 연습 1 정보

연습 1 정보


PTC Mathcad의 프로그래밍 연산자를 사용하여 부분 변수를 정의하고 값을 설정하고 프로그램에서 계산된 값을 구할 수 있을 뿐 아니라, 많은 횟수를 반복해야 하는 복잡한 계산을 수행할 수 있는 함수를 직접 정의할 수 있습니다.

이 연습을 마친 후에는 다음과 같은 작업을 수행할 수 있게 됩니다.

- 새 프로그램 만들기
- 부분 변수에 값 할당
- 고유한 함수 정의
- 프로그램 내에서 PTC Mathcad 연산자 및 함수 사용
- 프로그램 출력 계산
- 프로그램에서 계산된 값 구하기

[작업 1-1로 이동합니다.](#)

작업 1-1: 프로그램 작성

1. 새 프로그램을 삽입하려면 **수학** 탭의 **연산자 및 기호** 그룹에서 **프로그래밍**을 클릭합니다. 프로그래밍 연산자 목록이 열립니다. **프로그램** 연산자 를 클릭합니다. 프로그램 구조가 나타납니다.

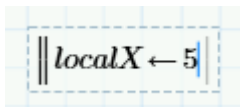


2. *localX*를 입력합니다.




부분 변수 *localX*는 프로그램 내부에서만 정의됩니다. 프로그램 외부에서는 이 변수를 참조할 수 없습니다.

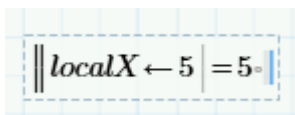
3. **수학** 탭의 **연산자 및 기호** 그룹에서 **프로그래밍**을 클릭합니다. 프로그래밍 연산자 목록이 열립니다. ← **부분 지정**을 클릭하고 5를 입력합니다.



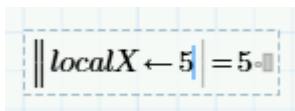
이것이 부분 지정 연산자입니다. 프로그램 내부에서는 정의 연산자가 적용되지 않습니다.

 프로그램 내부에서는 워크시트의 변수와 이름이 같은 부분 변수를 정의할 수 있지만 프로그램 외부에서는 그렇게 할 수 없습니다. PTC Mathcad에서는 이러한 변수를 서로 다른 두 변수로 취급합니다.

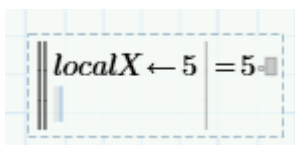
4. 프로그램을 계산하려면 **=**을 누릅니다. PTC Mathcad에 프로그램의 마지막 줄이 계산되어 표시됩니다.



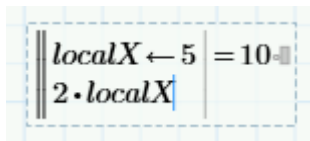
5. 커서를 지정 식의 끝에 배치합니다.




6. Enter 키를 누릅니다. 새 프로그램 줄이 나타납니다.



7. $2 * localX$ 를 입력하고 F5를 눌러 계산합니다. PTC Mathcad가 프로그램을 다시 계산합니다.



$$\left\| \begin{array}{l} localX \leftarrow 5 \\ 2 \cdot localX \end{array} \right\| = 10$$

 프로그램의 길이에 관계없이 결과는 항상 프로그램의 오른쪽 위에 표시됩니다.

8. 프로그램에서 계산된 마지막 식을 지정하려면 아래에 표시된 것처럼 프로그램을 변수에 지정합니다.

$$x := \left\| \begin{array}{l} localX \leftarrow 5 \\ 2 \cdot localX \end{array} \right\| = 10$$

9. 변수 $localX$ 및 x 를 계산합니다.

$$localX = ?$$

$$x = 10$$

x 와 달리 $localX$ 는 부분 변수이며 프로그램 외부에서는 변수 값을 알 수 없습니다.

[작업 1-2로 이동합니다.](#)

프로그래밍 자습서 > 작업 1-2: 함수 정의

작업 1-2: 함수 정의

1. 숫자가 양수인지 여부를 검사하는 함수를 정의합니다.

$$f(x) := \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

함수를 정의했으면 함수를 호출하고, 매개변수(예: 숫자, 변수)와 함께 사용할 수 있습니다.

2. 다른 값을 사용하여 함수를 호출하고 아래 그림과 같이 숫자가 양수인 경우 1을, 양수가 아닌 경우 0을 반환하는지 확인합니다.

$$f(5) = 1 \quad f(-5) = 0$$

3. 두 숫자의 최대 공약수와 차이를 계산하는 함수를 정의합니다.

$$f(x, y) := \begin{cases} a \leftarrow \gcd(x, y) \\ b \leftarrow |x - y| \\ [a \ b] \end{cases}$$

 프로그램 내에서 **gcd** 같은 모든 PTC Mathcad 기본 제공 함수를 사용할 수 있습니다.

4. $f(57, 48)$ 의 값을 확인합니다.

$$f(57, 48) = [3 \ 9]$$

함수는 요소가 두 개인 벡터로 출력됩니다. 첫 번째 숫자는 두 입력 숫자의 최대 공약수이고, 두 번째 숫자는 두 숫자의 차이입니다.

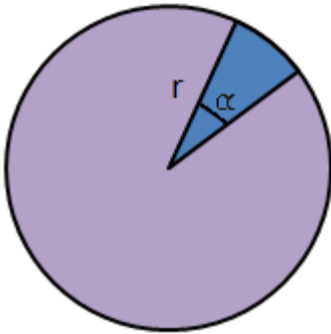
[작업 1-3으로 이동합니다.](#)

프로그래밍 자습서 > 작업 1-3: 연산자 사용

작업 1-3: 연산자 사용


연산자 및 단위 사용

PTC Mathcad 기본 제공 연산자를 사용하여 아래 표시된 원의 부채꼴 면적을 계산합니다.



1. 새 프로그램을 작성하고 부분 변수 r 을 1미터로 정의합니다.

$$\left| \begin{array}{l} r \leftarrow 1 \text{ m} \end{array} \right|$$

 PTC Mathcad는 문자 m을 단위로 인식합니다.

2. 각도 α 를 23도로 정의합니다.

$$\left| \begin{array}{l} r \leftarrow 1 \text{ m} \\ \alpha \leftarrow 23 \text{ deg} \end{array} \right|$$

3. 정적분 연산자를 삽입합니다.

$$\left| \begin{array}{l} r \leftarrow 1 \text{ m} \\ \alpha \leftarrow 23 \text{ deg} \\ \int d\theta \end{array} \right|$$

4. 아래와 같이 적분 자리 표시자에 필요한 값을 추가하고 적분에 $1/2$ 을 곱합니다.

$$\left| \begin{array}{l} r \leftarrow 1 \text{ m} \\ \alpha \leftarrow 23 \text{ deg} \\ \frac{1}{2} \int_0^\alpha r^2 d\theta \end{array} \right|$$

5. 프로그램을 계산하여 부채꼴 면적을 얻습니다.

$$\left\| \begin{array}{l} r \leftarrow 1 \text{ m} \\ \alpha \leftarrow 23 \text{ deg} \\ \frac{1}{2} \int_0^\alpha r^2 d\theta \end{array} \right\| = 0.201 \text{ m}^2$$

PTC Mathcad 기본 제공 함수 및 행렬 사용

1. 아래와 같이 새 프로그램을 작성하고 벡터 두 개를 정의합니다.

$$x := \left\| \begin{array}{l} m1 \leftarrow \begin{bmatrix} 4 & 11 \\ 5 & 8 \end{bmatrix} \\ m2 \leftarrow \begin{bmatrix} 3 & 6 \\ 12 & 7 \end{bmatrix} \end{array} \right\|$$

2. 함수 탭의 통계 목록에서 **mean** 함수를 삽입합니다. 함수와 빈 자리 표시자가 추가됩니다.

$$x := \left\| \begin{array}{l} m1 \leftarrow \begin{bmatrix} 4 & 11 \\ 5 & 8 \end{bmatrix} \\ m2 \leftarrow \begin{bmatrix} 3 & 6 \\ 12 & 7 \end{bmatrix} \\ \text{mean}(\text{ }, \text{ }, \text{ }, \text{ }) \end{array} \right\|$$

3. $m1$ 및 $m2$ 요소의 평균을 계산하고 나머지 자리 표시자를 삭제합니다.

$$x := \left\| \begin{array}{l} m1 \leftarrow \begin{bmatrix} 4 & 11 \\ 5 & 8 \end{bmatrix} \\ m2 \leftarrow \begin{bmatrix} 3 & 6 \\ 12 & 7 \end{bmatrix} \\ \text{mean}(m1, m2) \end{array} \right\| = 7$$

실습

다음 연습으로 이동하기 전에 벡터 v 를 입력으로 사용하는 함수 f 를 정의합니다.

함수 f 는 PTC Mathcad 기본 제공 함수를 사용하는 프로그램으로, 벡터의 길이, 벡터 v 의 최대값 요소 및 벡터 v 의 중앙값을 포함한 세 개의 요소로 이루어진 벡터를 반환합니다.

[연습 2로 이동합니다.](#)

프로그래밍 자습서 > 연습 2 정보

연습 2 정보

PTC Mathcad에서 프로그램 내부에 분기를 정의할 수 있습니다. 각 분기에서 지정한 작업은 특정 조건이 충족되는 경우에만 수행됩니다.

이 연습을 마친 후에는 다음과 같은 작업을 수행할 수 있게 됩니다.

- *if* 문 작성
- *if-else* 문 작성
- 프로그램에서 분기 사용

[작업 2-1로 이동합니다.](#)

작업 2-1: If 문 작성

인수의 값을 "BLACK"에서 "WHITE"로 또는 그 반대로 수정하는 함수 *reverse*를 작성합니다.

1. 변수 *var*을 입력 인수로 사용하는 새 함수 *reverse*를 생성합니다.

$$\text{reverse}(var) := \left| \right|$$

2. *if* 문을 추가하려면 수학 탭의 연산자 및 기호 그룹에서 **프로그래밍**을 클릭합니다. 프로그래밍 연산자 목록이 열립니다. **if**를 클릭합니다.

$$\text{reverse}(var) := \left| \left| \begin{array}{l} \text{if } \left| \right| \end{array} \right| \right|$$


"if"와 같은 프로그래밍 연산자의 이름을 입력한 다음 Ctrl+J를 누르면 모든 자리 표시자가 있는 프로그래밍 연산자로 이름을 변환할 수 있습니다.

3. 아래와 같이 *var*의 값이 "BLACK"이면 함수가 값 "WHITE"를 반환합니다. 비교에 사용할 "같음" 부울 연산자를 추가하려면 수학 탭의 연산자 및 기호 그룹에서 연산자를 클릭합니다. 연산자 목록이 열립니다. = 같음을 클릭합니다.

$$\text{reverse}(var) := \left| \left| \begin{array}{l} \text{if } var = \text{"BLACK"} \\ \left| \right| \text{"WHITE"} \end{array} \right| \right|$$

4. 다른 *if* 문을 추가하려면 아래와 같이 커서를 배치하고 Enter 키를 누릅니다.

$$\text{reverse}(var) := \left| \left| \begin{array}{l} \text{if } var = \text{"BLACK"} \\ \left| \right| \text{"WHITE"} \\ \left| \right| \end{array} \right| \right|$$

커서가 "WHITE" 바로 오른쪽에 있는 상태에서 Enter 키를 누르면 PTC Mathcad가 *if* 블록 내에 줄을 추가합니다.

5. 아래와 같이 반대 경우를 지정합니다.

$$\text{reverse}(var) := \left| \left| \begin{array}{l} \text{if } var = \text{"BLACK"} \\ \left| \right| \text{"WHITE"} \\ \text{if } var = \text{"WHITE"} \\ \left| \right| \text{"BLACK"} \end{array} \right| \right|$$


PTC Mathcad는 중첩 *if* 문을 지원합니다.

6. 아래와 같이 *reverse*를 호출하여 변수 *myvar*의 값을 변경합니다.

$$\text{myVar} := \text{"BLACK"} \quad \text{myVar} := \text{reverse}(\text{myVar}) \quad \text{myVar} = \text{"WHITE"}$$

파일이 열린 상태로 [작업 2-2](#)로 이동합니다.


작업 2-2: If-Else If 문 작성

elseif 또는 *else* 문을 사용하면 특정 *if* 문이 true인지 아니면 false인지에 따라 식을 계산할 수 있습니다.

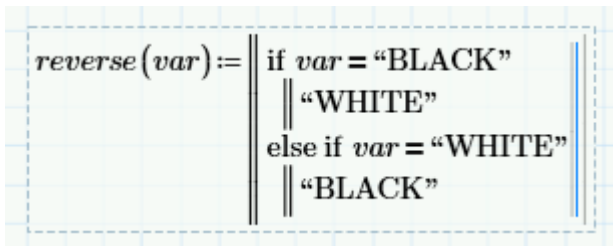
1. 작업 2-1에서 작성한 함수에서 두 번째 *if* 문을 *else if*로 수정하기 위해 두 번째 *if* 프로그래밍 연산자를 선택하고 수학 탭의 연산자 및 기호 그룹에서 프로그래밍을 클릭한 다음 **else if**를 클릭합니다.

```
reverse(var) := 
$$\begin{cases} \text{if } var = \text{"BLACK"} \\ \quad \text{"WHITE"} \\ \text{else if } var = \text{"WHITE"} \\ \quad \text{"BLACK"} \end{cases}$$

```

 *else if* 연산자는 *if* 또는 *else if* 문 바로 다음에만 사용할 수 있습니다.

2. *else* 문을 추가하려면 아래와 같이 다른 모든 옵션을 포함하는 위치에 커서를 배치합니다.



```
reverse(var) := 
$$\begin{cases} \text{if } var = \text{"BLACK"} \\ \quad \text{"WHITE"} \\ \text{else if } var = \text{"WHITE"} \\ \quad \text{"BLACK"} \end{cases}$$

```

3. 빈 *else* 문을 추가하려면 수학 탭의 연산자 및 기호 그룹에서 프로그래밍을 클릭한 다음 **else**를 클릭합니다.

```
reverse(var) := 
$$\begin{cases} \text{if } var = \text{"BLACK"} \\ \quad \text{"WHITE"} \\ \text{else if } var = \text{"WHITE"} \\ \quad \text{"BLACK"} \\ \text{else} \\ \end{cases}$$

```

4. "RAINBOW"를 입력합니다.

```
reverse(var) := 
$$\begin{cases} \text{if } var = \text{"BLACK"} \\ \quad \text{"WHITE"} \\ \text{else if } var = \text{"WHITE"} \\ \quad \text{"BLACK"} \\ \text{else} \\ \quad \text{"RAINBOW"} \end{cases}$$

```


5. *reverse*를 호출하여 5의 값을 구합니다.

```
reverse(5) = "RAINBOW"
```

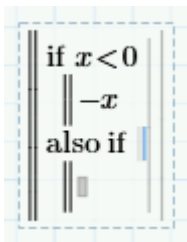
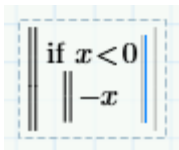
실습

1. 다음 연습으로 이동하기 전에 매개변수 x 가 인수이고 *also if*를 사용하여 다음과 같은 작업을 수행하는 함수 f 를 작성합니다.

- x 가 문자열이면 부분 변수 y 에 1을 지정합니다. **IsString** 함수를 사용합니다.
- x 의 길이가 5보다 크면 y 에 2를 지정합니다. **strlen** 함수를 사용합니다.
- 그렇지 않으면 y 에 3을 지정합니다.

 *also if* 문은 *if* 문 또는 *also if* 문 다음에만 삽입할 수 있습니다. *if* 문이 true이면 모든 *also if* 문이 계산됩니다.

also if 문을 추가하기 위해 *if* 블록의 마지막 줄 끝에 커서를 배치하고(아래 참조) 수학 탭의 연산자 및 기호 그룹에서 **프로그래밍**을 클릭합니다. 프로그래밍 연산자 목록이 열립니다. **also if**를 클릭합니다.



2. 워크시트에서 다음과 같은 함수를 정의합니다.

$$f(x) := x^2 - 1$$

매개변수 x 가 인수이고 아래 설명과 같이 동작하는 함수 h 를 작성합니다.

함수 h 는 조각 함수를 포함하는 프로그램입니다.

x 값이 -1에서 1 사이이면 $h(x)$ 값은 워크시트의 앞부분에 정의한 $f(x)$ 이고 그렇지 않으면 $h(x)$ 값은 $f(x)$ 곱하기 -1입니다.

XY 도표를 추가하여 함수를 도표화합니다.

[연습 3으로 이동합니다.](#)

프로그래밍 자습서 > 연습 3 정보

연습 3 정보

PTC Mathcad에서는 *for* 루프를 정의하여 같은 식을 여러 번 반복할 수 있으며, 조건이 충족될 때까지 몇 번 반복해야 하는지 모를 경우 유용하게 사용할 수 있는 *while* 루프를 정의할 수도 있습니다. 이 연습을 마친 후에는 다음과 같은 작업을 수행할 수 있게 됩니다.

- *for* 루프 작성
- *while* 루프 작성
- *continue*를 사용하여 루프 반복 건너뛰기
- *break*를 사용하여 루프 종료
- Try/On Error 문 사용

[작업 3-1로 이동합니다.](#)

작업 3-1: For 루프 작성

단일 식을 사용하여 벡터의 여러 요소를 수정하려면 *for* 루프를 사용합니다.


1. 변수 *vec*가 인수인 함수 *f*를 정의하고 새 프로그램을 작성합니다.
2. 빈 *for* 루프를 정의하려면 수학 탭의 연산자 및 기호 그룹에서 프로그래밍을 클릭한 다음 **for**를 클릭합니다.

$$f(\text{vec}) := \left\| \begin{array}{l} \text{for } \square \in \square \\ \square \end{array} \right\|$$

 연산자 \in (다음에 속함)를 수정할 수 없습니다. 이 연산자는 *for* 문 구문의 일부입니다.

3. 아래와 같이 반복 변수 *i*와 값 범위 (0..2)를 지정합니다.

$$f(\text{vec}) := \left\| \begin{array}{l} \text{for } i \in 0..2 \\ \square \end{array} \right\|$$

-  • 값 범위는 단일 값, 벡터 또는 값 범위를 정의하는 행렬일 수 있습니다.
- PTC Mathcad에서 벡터 또는 행렬의 기본 원점은 0입니다.

4. 아래 그림과 같이 변수 *i* 값을 벡터의 첫 세 요소에 지정합니다.

$$f(\text{vec}) := \left\| \begin{array}{l} \text{for } i \in 0..2 \\ \begin{array}{l} \text{vec}_i \leftarrow i \\ \text{vec} \end{array} \end{array} \right\|$$

5. *vec*에 문자열 행 두 개가 있을 때 함수 *f*를 계산합니다.

$$f\left(\begin{bmatrix} \text{"a"} \\ \text{"b"} \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

결과는 변수 *i*에 지정된 대로 행이 세 개인 벡터입니다. 원래 두 개의 행이 *for* 루프에 의해 초과로 생성되었습니다.

6. *vec*에 행 세 개가 있을 때 함수 *f*를 계산합니다.

$$f\left(\begin{bmatrix} \text{"a"} \\ \text{"b"} \\ \text{"c"} \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

결과는 변수 *i*에 지정된 대로 행이 세 개인 벡터입니다. 원래 세 개의 행이 *for* 루프에 의해 초과로 생성되었습니다.

7. *vec*에 행 다섯 개가 있을 때 함수 *f*를 계산합니다.

$$f\left(\begin{bmatrix} \text{"a"} \\ \text{"b"} \\ \text{"c"} \\ \text{"d"} \\ \text{"e"} \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \\ 2 \\ \text{"d"} \\ \text{"e"} \end{bmatrix}$$

결과는 행이 다섯 개인 벡터이며, 처음 세 행은 *for* 루프에 의해 생성되었고 마지막 두 행은 변경되지 않은 상태로 유지됩니다.

[작업 3-2로 이동합니다.](#)

작업 3-2: While 루프 작성

식 복사

기본 While 함수 작성

0부터 n 까지의 모든 숫자를 합하는 함수 σ 를 작성합니다.

1. 변수 n 이 인수인 함수 σ 를 정의하고 새 프로그램을 작성합니다.


$$\sigma(n) := \parallel$$

2. 빈 **while** 루프를 추가하려면 수학 탭의 연산자 및 기호 그룹에서 **프로그래밍**을 클릭한 다음 **while**을 클릭합니다.

$$\sigma(n) := \parallel \text{while } \parallel$$

3. $n > 0$ 인 동안 **while** 루프가 계속 실행되고 루프 내에서 1씩 n 을 감소시키도록 지정합니다.

$$\sigma(n) := \parallel \text{while } n > 0 \parallel$$


 **for** 루프와 달리 **while** 반복 변수를 직접 증가시키거나 감소시켜야 합니다.

4. 현재 반복 변수 값을 합계에 더하기 위해 **while** 문 바로 다음에 아래와 같은 줄을 입력합니다.

$$\sigma(n) := \parallel \text{while } n > 0 \parallel$$

5. σ 의 값을 반환합니다.

$$\sigma(n) := \parallel \text{while } n > 0 \parallel$$

 마지막으로 반복 변수를 업데이트합니다. 그렇지 않으면 첫 번째 반복 변수 더하기를 삭제해야 합니다.

6. 5에 대해 σ 의 값을 계산합니다.

$$\sigma(5) = 15$$

예상대로 프로그램은 다음 합계와 동일합니다.

$$\sum_{i=1}^5 i = 15$$

식 복사

Continue 문 추가

루프를 계속 실행하지만 특정 반복을 건너뛰려면 *continue* 문을 추가합니다.

0부터 n 까지에서 17로 나뉘지는 숫자를 제외한 모든 숫자를 합하는 함수를 작성합니다.

1. 위 함수를 복사하고 함수 이름을 *sigma_not17*로 바꿉니다.

```
sigma_not17(n) := || while n > 0
                  || || sum ← sum + n
                  || || n ← n - 1
                  || sum
```

2. *while* 루프 안에서 *while* 문 아래에 새 줄을 추가합니다.

```
sigma_not17(n) := || while n > 0
                  || ||
                  || || sum ← sum + n
                  || || n ← n - 1
                  || sum
```

3. *if* 문을 추가하고 아래에 식을 입력합니다.

```
sigma_not17(n) := || while n > 0
                  || || if mod(n, 17) = 0
                  || || ||
                  || || || sum ← sum + n
                  || || || n ← n - 1
                  || sum
```

4. 무한 루프를 방지하기 위해 n 을 1씩 감소시킵니다.

5. *continue* 문을 추가하려면 수학 탭의 연산자 및 기호 그룹에서 **프로그래밍**을 클릭한 다음 **continue**를 클릭합니다.

```
sigma_not17(n) := || while n > 0
                  || || if mod(n, 17) = 0
                  || || || n ← n - 1
                  || || || continue
                  || || || sum ← sum + n
                  || || || n ← n - 1
                  || sum
```

6. 16과 17에 대해 *sigma_not17*의 값을 계산합니다.

$$\text{sigma_not17}(16) = 136$$

$$\sigma_{not17}(17) = 136$$

Break 문 추가

식 복사

모든 숫자를 합하고 카운터가 20보다 크면 루프를 종료하는 프로그램을 작성합니다.

1. 변수 *sum*을 정의하고 새 프로그램을 작성합니다.


$$sum := \left| \right|$$

2. 빈 *while* 루프를 추가하려면 수학 탭의 연산자 및 기호 그룹에서 프로그래밍을 클릭한 다음 **while**을 클릭합니다.

$$sum := \left| \begin{array}{l} \text{while } () \\ \left| \right| \end{array} \right|$$

3. *while* 루프가 무한 실행되도록 지정합니다.

$$sum := \left| \begin{array}{l} \text{while } (1) \\ \left| \right| \end{array} \right|$$

 *while* 루프는 괄호 안의 식 값이 0이 아닌 한 계속 실행됩니다.

4. *sum* 및 *i*를 초기화합니다.


$$sum := \left| \begin{array}{l} sum \leftarrow 0 \\ i \leftarrow 0 \\ \text{while } (1) \\ \left| \right| \end{array} \right|$$

5. 루프 안에서 반복 변수 *i*의 값을 변수 *sum*에 지정하고 *i*를 1씩 증가시킵니다.

$$sum := \left| \begin{array}{l} sum \leftarrow 0 \\ i \leftarrow 0 \\ \text{while } (1) \\ \left| \begin{array}{l} sum \leftarrow sum + i \\ i \leftarrow i + 1 \end{array} \right| \end{array} \right|$$

6. *sum*의 값을 반환합니다.

$$sum := \left| \begin{array}{l} sum \leftarrow 0 \\ i \leftarrow 0 \\ \text{while } (1) \\ \left| \begin{array}{l} sum \leftarrow sum + i \\ i \leftarrow i + 1 \end{array} \right| \\ sum \end{array} \right|$$

 현재 이 루프는 무한 루프입니다.

7. 루프를 중단하기 위해 $if\ i > 20$ 을 입력하고 *break* 문을 추가합니다. *break* 문을 추가하려면 수학 탭의 연산자 및 기호 그룹에서 프로그래밍을 클릭한 다음 **break**를 클릭합니다.

식 복사


```
sum :=
|
| sum ← 0
| i ← 0
| while (1)
|   |
|   | if (i > 20)
|   |   |
|   |   | break
|   |   sum ← sum + i
|   |   i ← i + 1
| sum
```

8. *sum*을 계산합니다.

sum = 210


9. 루프를 중단하고 프로그램을 종료하려면 *break* 문을 선택하고 수학 탭의 연산자 및 기호 그룹에서 프로그래밍을 클릭한 다음 **return**을 클릭하여 *break* 문을 *return* 문으로 수정합니다. 아래와 같이 자리 표시자에 *sum*을 입력합니다.

```
sum :=
|
| sum ← 0
| i ← 0
| while (1)
|   |
|   | if (i > 20)
|   |   |
|   |   | return sum
|   |   sum ← sum + i
|   |   i ← i + 1
| sum
```

 프로그램을 즉시 종료하려면 *return*을 사용합니다.

실습

다음 작업으로 이동하기 전에 *while* 루프를 사용하여 계승 함수를 구현하는 함수 *fact(n)*를 작성합니다. *n*이 1보다 큰 동안 계속 실행되는 루프를 정의합니다. 루프 안에서 *n*에 변수 *product*을 곱하고 (계승 결과 저장) *n*을 1씩 감소시킵니다.

 PTC Mathcad에서 프로그램 변수는 기본적으로 0으로 설정됩니다. 프로그램의 시작 부분에서 *product*에 1을 지정해야 합니다. 그렇지 않으면 프로그램이 모든 인수에 대해 0을 반환합니다.

작업 3-3으로 이동합니다.

작업 3-3: Try-On-Error 문 작성

*try-on-error*를 사용하여 프로그램을 실행하는 동안 오류가 발생할 경우 수행해야 할 작업을 지정합니다. 예를 들어 0으로 나누기는 오류입니다.

1. 함수 $f(x)$ 를 정의합니다.

$$f(x) := \square$$

2. *try/on error* 문을 삽입하려면 수학 탭의 연산자 및 기호 그룹에서 **프로그래밍**을 클릭한 다음 **try**를 클릭합니다.

$$f(x) := \begin{array}{|l} \text{try} \\ \hline \text{on error} \end{array}$$

3. $1/2-x$ 를 입력합니다.

$$f(x) := \begin{array}{|l} \text{try} \\ \hline \frac{1}{2-x} \\ \text{on error} \end{array}$$

try 블록은 오류가 발생하지 않을 때 PTC Mathcad가 수행하는 함수로, 여러 개의 식을 포함할 수 있습니다.

4. 커서를 *on error* 블록 안으로 이동하고 수학 탭의 연산자 및 기호 그룹에서 상수를 클릭한 다음 무한대(∞)를 선택합니다.

$$f(x) := \begin{array}{|l} \text{try} \\ \hline \frac{1}{2-x} \\ \text{on error} \\ \hline \infty \end{array}$$

5. 아래 그림과 같이 다양한 x 값에 대해 $f(x)$ 를 계산합니다.

$$f(1)=1 \quad f(2)=1 \cdot 10^{307} \quad f(-2)=0.25$$

축하합니다! 프로그래밍 자습서를 완료했습니다.